University of Zurich[UZH]

Zurich Open Repository and Archive
University of Zurich
University Library
Strickhofstrasse 39
CH-8057 Zurich
www.zora.uzh.ch

Year: 2020

Dynamic obstacle avoidance for quadrotors with event cameras

Falanga, Davide ; Kleber, Kevin ; Scaramuzza, Davide

Abstract: Today's autonomous drones have reaction times of tens of milliseconds, which is not enough for navigating fast in complex dynamic environments. To safely avoid fast moving objects, drones need low-latency sensors and algorithms. We departed from state-of-the-art approaches by using event cameras, which are bioinspired sensors with reaction times of microseconds. Our approach exploits the temporal information contained in the event stream to distinguish between static and dynamic objects and leverages a fast strategy to generate the motor commands necessary to avoid the approaching obstacles. Standard vision algorithms cannot be applied to event cameras because the output of these sensors is not images but a stream of asynchronous events that encode per-pixel intensity changes. Our resulting algorithm has an overall latency of only 3.5 milliseconds, which is sufficient for reliable detection and avoidance of fast-moving obstacles. We demonstrate the effectiveness of our approach on an autonomous quadrotor using only onboard sensing and computation. Our drone was capable of avoiding multiple obstacles of different sizes and shapes, at relative speeds up to 10 meters/second, both indoors and outdoors.

DOI: https://doi.org/10.1126/scirobotics.aaz9712

Posted at the Zurich Open Repository and Archive, University of Zurich
ZORA URL: https://doi.org/10.5167/uzh-197705
Journal Article
Accepted Version

Originally published at:
Falanga, Davide; Kleber, Kevin; Scaramuzza, Davide (2020). Dynamic obstacle avoidance for quadrotors with event cameras. Science Robotics, 5(40):eaaz9712.
DOI: https://doi.org/10.1126/scirobotics.aaz9712

This paper has been accepted for publication at Science Robotics,
Volume 5, Issue 40, March 18, 2020.
DOI: https://doi.org/10.1126/scirobotics.aaz9712

Low Latency Avoidance of Dynamic Obstacles for Quadrotors with Event Cameras

Davide Falanga, Kevin Kleber, Davide Scaramuzza
Depts. of Informatics & Neuroinformatics, ETH and University of Zurich

In this paper, we address one of the fundamental challenges for micro aerial vehicles: dodging fast-moving objects using only onboard sensing and computation. Effective avoidance of moving obstacles requires fast reaction times, which entails low-latency sensors and algorithms for perception and decision making. All existing works rely on standard cameras, which have an average latency of tens of milliseconds and suffer from motion blur. We depart from state of the art by relying on a novel bioinspired sensor, called event camera, with reaction times of microseconds, which perfectly fits our task requirements. However, because the output of this sensor is not images but a stream of asynchronous events that encode per-pixel intensity changes, standard vision algorithms cannot be applied. Thus, a paradigm shift is necessary to unlock the full potential of event cameras. Our proposed framework exploits the temporal information contained in the event stream to distinguish between static and dynamic objects, and makes use of a fast strategy to generate the motor commands necessary to avoid the detected obstacles. Our resulting algorithm has an overall latency of only $3.5\,\mathrm{ms}$, which is sufficient for reliable detection and avoidance of fast-moving obstacles. We demonstrate the effectiveness of our approach on an autonomous quadrotor avoiding multiple obstacles of different sizes and shapes, at relative speeds up to $10\,\mathrm{m/s}$, both indoors and outdoors.

Figure 1: Sequence of an avoidance maneuver.

Videos of the Experiments

All the experiments reported in this manuscript is available at `http://rpg.ifi.uzh.ch/event_based_avoidance`

Introduction

Micro aerial vehicles (MAVs) are at the forefront of this century's technological shift. They are becoming ubiquitous, giving birth to new, potentially disruptive markets worth several billion dollars, such as aerial imaging (forecast value of 4 billion USD by 2025 (1)), last-mile delivery (90 billion USD by 2030 (2)) and aerial mobility (almost 8 billion USD in 2030 (3)).

Keeping a vehicle airborne above a crowd poses large safety risks. Several drone crashes have been recently reported in the news, due to either objects tossed at quadrotors during public events (4, 5), or collisions with birds (6, 7). Enabling MAVs to evade fast-moving objects (cf. Fig. 1) is therefore critical for the deployment of safe flying robots on a large scale and is still unsolved.

The Challenge

The temporal latency between perception and action plays a key role in obstacle avoidance. The higher the latency, the lower the time the robot has to react and execute an avoidance maneuver (8). This is especially critical for MAVs, where a collision can not only damage the environment, but also cause severe hardware failure. Additionally, micro quadrotors have reduced payload capabilities, which puts a hard bound on the sensing and computing resources they can carry.

The existing literature on obstacle avoidance for MAVs relies on standard cameras (in a monocular (9–11) or stereo configuration (12–15)) or on depth cameras (16–18). However, these works assume that the obstacles in the environment are either static or quasi-static (i.e., slow relative motion).

Similarly, state-of-the-art consumer drones are not currently capable to reliably detect and avoid moving obstacles. For example, the Skydio drone, as of today one of the most advanced autonomous drones on the market, is not capable of dealing with moving objects (*If you throw a ball at it, it's almost certainly not going to get out of the way*, said Adam Bry, CEO of Skydio (19)).

Developing effective algorithms to avoid dynamic obstacles is therefore a key challenge in robotics research, as well as a highly admired goal by major industry players.

Event Cameras

To avoid fast-moving obstacles, we need to perceive them fast. As it turns out, standard cameras are not good enough: they have an average latency of tens of milliseconds (the exposure time of

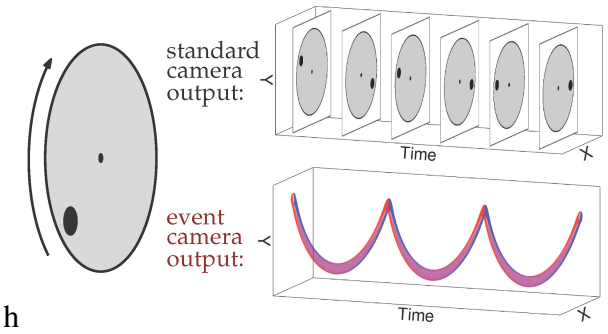


Figure 2: Comparison of the output of a conventional camera and that of an event camera when they are looking at a rotating disk with a black dot. While a conventional camera captures frames at a fixed rate, an event camera only outputs the sign of brightness changes continuously in the form of a spiral of events in space-time (red: positive changes, blue: negative changes).

a standard camera varies between 1 and 100 ms). Therefore, their limitations arise from their physical nature and cannot be solved with sophisticated algorithms. The solution is given by a novel type of sensor, called event camera, which has reaction times of microseconds. In a recent study on the role of perception latency for high-speed sense and avoid (8), it was shown analytically that, using event cameras, the latency between the time a visual signal is triggered by the sensor and processed to output control commands is significantly lower than that of standard cameras (ms vs tens of ms). This promises terrific consequences for robotics applications. However, because the output of an event camera is not images but a stream of asynchronous events, standard vision algorithms cannot be applied. Thus, novel algorithms need to be devised to unlock the full potential of event cameras for the task at hand.

Event cameras (20) are bio-inspired sensors that work radically different from traditional cameras. Instead of capturing images at a fixed rate, an event camera measures per-pixel brightness changes asynchronously. This results in a stream of events at microsecond resolution. More specifically, an event camera has smart pixels that trigger information *independently* of each other: whenever a pixel detects a change of intensity in the scene (e.g., caused by relative motion), that pixel will trigger an information at the time the intensity change was detected. This information is called *event*, and encodes the time (at microsecond resolution) at which the event occurred, the pixel location, and the sign of the intensity changes. Let $t_{\text{k-1}}$ be the last time when an event fired at a pixel location $\mathbf{x}$, and let $L_{\text{k-1}} = L\left(\mathbf{x}, t_{\text{k-1}}\right)$ be the intensity level at such pixel at time $t_{\text{k-1}}$. A new event is fired at the same pixel location at time t_{k} as soon as the difference between the intensity $L_{\text{k-1}}$ and L_{k} is larger than a user-define threshold $C > 0$. In other words, an event is fired if $\|L\left(\mathbf{x}, t_{\text{k}}\right) - L\left(\mathbf{x}, t_{\text{k-1}}\right)\| > C$ (positive event) or $\|L\left(\mathbf{x}, t_{\text{k}}\right) - L\left(\mathbf{x}, t_{\text{k-1}}\right)\| < -C$ (negative event). We refer the reader to (21) for further details.

To better highlight what happens across the entire sensor, we compare the output of an event camera to the one of a conventional camera in Fig. 2 and in a video (22).

Event cameras can thus be seen as *asynchronous*, *motion-activated* sensors, since they

provide measurements only *if* and *where* there is relative motion between the camera and the environment. And because their latency is in the order of microseconds, they are a natural choice for detection and avoidance of fast moving obstacles by flying MAVs.

If one removes the events induced by the ego-motion of the vehicle (23, 24), one can directly obtain information about the moving part of the scene. This leads to multiple advantages over standard cameras for detection of dynamic obstacles: (i) the output is sparser and lighter than a frame, therefore cheaper to process; (ii) no segmentation between static and dynamic objects is necessary, since to do so it is possible to exploit the temporal statistic of each event; (iii) their high temporal resolution (in the order of microseconds) allows low-latency sensing.

Related Work

In recent years, event cameras have attracted the interest of the robotics community (21). Obstacle detection is among the applications with the highest potential, and previous works investigated the use of these sensors to detect collisions (25) and track objects (23, 26). However, very few examples of closed-loop control based on event cameras are available in the literature. Among these, the majority of the works focuses on simple, low-dimensional tasks, such as 1-DoF heading regulation (27, 28), stereo-camera gaze control (29, 30), 2-DoF pole balancing (31), 1-DoF robotic goalkeeping (32, 33), or navigating ground robots among static obstacles (34–36).

Examples of closed-loop control of more complex robotic systems, such as quadrotors, using event cameras are our recent works (37–39). In (37), we proposed an event-based visual-inertial odometry algorithm for state estimation and closed-loop trajectory tracking of a quadrotor. Instead, (38) and (39) are the most related to this paper. In (38), we analyzed the feasibility of detecting spherical objects thrown at a *stationary* event camera on small embedded processors for quadrotors. In (39), we showed preliminary results in using shallow neural networks for segmenting moving objects from event streams and demonstrated an application to quadrotor obstacle avoidance. However, the resulting sensing latency was $60\,\mathrm{ms}$ rather than the $3.5\,\mathrm{ms}$ of this paper, thus strongly limiting the maximum relative speed at which moving objects could be evaded. Additionally, differently from this paper, there we did not consider the relative distance and velocity to compute the avoidance commands. To the best of our knowledge, this is the first work that implements and demonstrates low latency ($3.5\,\mathrm{ms}$) dynamic obstacle dodging on an autonomous quadrotor with relative speeds up to $10\,\mathrm{m/s}$.

Overview of the Approach and Contributions

Our moving-obstacle detection algorithm works by collecting events during a short-time sliding window and compensating for the motion of the robot within such a time window. Fig. 3 shows the effects of the ego-motion compensation: on the left side, the 3D volume of the events accumulated during an arbitrary time window of $10\,\mathrm{ms}$, on the right side the same events, after ego-motion compensation, back-projected on the image plane. We analyze the temporal

Class	**Name**	**Reaction Times** [ms]	**Reference**
Bird	Starling (*Sturnus Vulgaris*)	76 ± 38	(40)
Bird	Pigeon (*Columba livia domestica*)	$50 - 100$	(41)
Bird	Hummingbird (*Eugenes fulgens*)	$20 - 50$	(42)
Bird	Yellowhammer (*Emberiza citrinella*)	$\sim$ 10	(43)
Bird	Greenfinch (*Carduelis chloris*)	$\sim$ 30	(43)
Fish	Squid (*Loligo opalescens*)	$50 - 75$	(44)
Fish	Herring Larvae	$\sim$ 40	(45)
Fish	Crab (*Chasmagnathus granulatus*)	$30 - 50$	(46)
Fish	Goldfish (*Carassius auratus*)	$\sim$ 150	(47)
Fish	Larval Zebrafish	< 100	(48)
Fish	Coral Reef Fish (*Spiny Chromis*)	$\sim$ 15	(49)
Insect	House fly (*Musca domestica*)	$30 - 50$	(50)
Insect	Condylostylus	$2 - 5$	(51)
Insect	Skipper butterfly (*Hesperiidae*)	< 17	(51)
Insect	Cockroach (*Periplaneta Americana*)	$\sim$ 50	(51)
Insect	Hawkmoth (*Manduca sexta*)	$50 - 100$	(52)
Insect	Fruit fly (*Drosophila*)	61 ± 21	(53)
Insect	Locust (*Locusta migratoria migratoriode*)	$\sim$ 50	(54)
Mammal	Shrew	$\sim$ 10	(55)
Mammal	Mouse	$\sim$ 20	(55)
Mammal	Rat	$\sim$ 30	(55)
Mammal	Rabbit	$\sim$ 40	(55)
Mammal	Cat	$\sim$ 30	(55)
Mammal	Guinea Pig	$\sim$ 45	(55)
Mammal	Dog	$\sim$ 40	(55)
Mammal	Sheep	$\sim$ 60	(55)
Mammal	Goat	$\sim$ 40	(55)
Mammal	Giraffe	$\sim$ 100	(55)
Mammal	Elephant	$\sim$ 100	(55)
Mammal	Human	$\sim$ 200	(56)

Table 1: Reaction times to visual stimula in animals. The estimated reaction times are reporting in different formats (i.e., confidence intervals, ranges, upper-bounds or order of magnitude) based on the information provided in the literature.

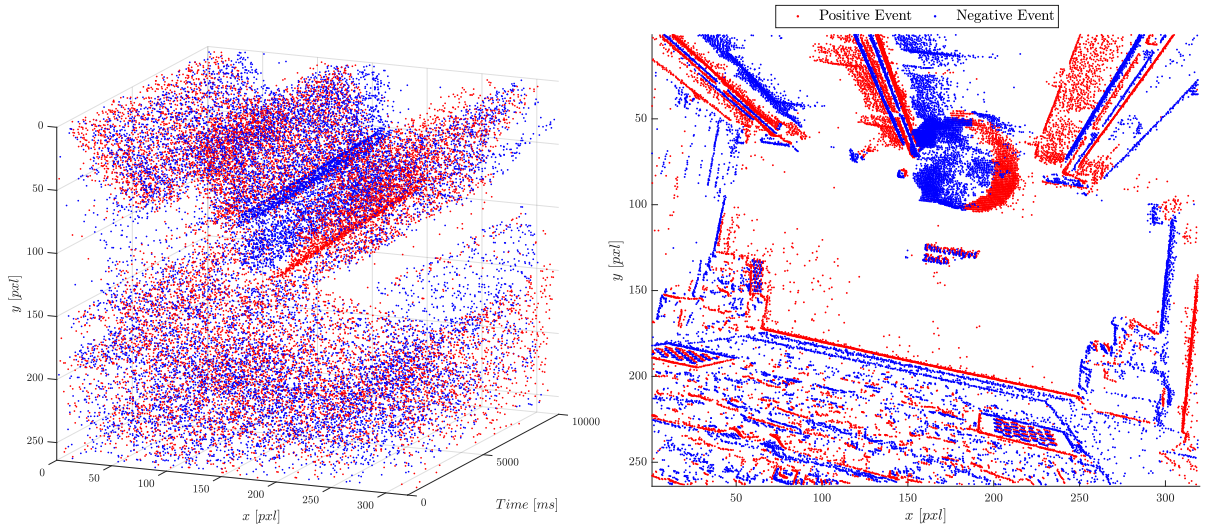
statistics of the motion-compensated events to remove those generated by the static part of the environment.

Broadly speaking, our algorithm is based on the intuition that the static part of the scene fires events uniformly across the entire time window and, after the ego-motion compensation, the pixels belong to show a uniform distribution of timestamps; conversely, dynamic objects generate ego-motion compensated events that are accumulated around specific sections of the time window, and can, therefore, be distinguished. Our technique to detect moving obstacle using event cameras is based on the method proposed in (23), where the authors used an optimization-based ego-motion compensation scheme. We modified that method to only rely on an Inertial Measurement Unit for the ego-motion compensation, without using any optimization scheme, which makes the algorithm sufficiently fast to run in real-time on a small on-board computer. An analysis of the impact of neglecting the linear component of the robot's ego-motion is provided in Sec. Impact of Using a Simplified Ego-Motion Estimation Algorithm of the supplementary materials. An intuitive explanation of how and why our algorithm works is provided in Sec. Time Statistics of Events to Detect Moving Obstacles of the supplementary material, while we refer the reader to Sec. Ego-Motion Compensation of the Events for a detailed explanation of this process, which allows us to obtain a so-called *event frame*, containing only events coming from moving objects, at very high rate. We leverage a fast clustering algorithm to tell apart different objects in the event frame and use a Kalman filter to obtain information about their velocity. Fig. 4 provides a visual explanation of the steps involved in our algorithm, which is thoroughly described in Sec. Obstacle Detection.

The position and velocity of each obstacle relative to the camera are then fed to a fast avoidance algorithm designed to leverage the low sensing latency. To do so, we use a reactive avoidance scheme based on the use of artificial potential fields (57) relying on fast geometric primitives to represent the obstacles, which renders it computationally inexpensive. We propose a novel formulation of the repulsive field, which better suits the task of avoiding fast-moving obstacles by taking into account the need for a prompt reaction of the robot when an obstacle is detected. Compared to previous approaches, our formulation of the repulsive potential increases significantly faster as the distance between the robot and the obstacle decreases in order to render the avoidance maneuver more reactive and agile. Additionally, we consider both the magnitude and the direction of the obstacle's velocity to decide in which direction to evade, and introduce a decay factor in the magnitude of the potential to take into account that the obstacles we consider are dynamic, i.e., they do not occupy the same position in time. Further details about the avoidance strategy are available in Sec. Obstacle Avoidance.

Our approach prioritizes computation speed over accuracy, therefore, we trade-off detection accuracy for latency. Nevertheless, in Sec. Accuracy and Success Rate we show that our algorithm only takes on average $3.5\,\mathrm{ms}$ (from the moment it receives the events to process to when it sends the first command to avoid the detected obstacles) to detect moving obstacles with a position error usually in the order of a few tens of centimeters.

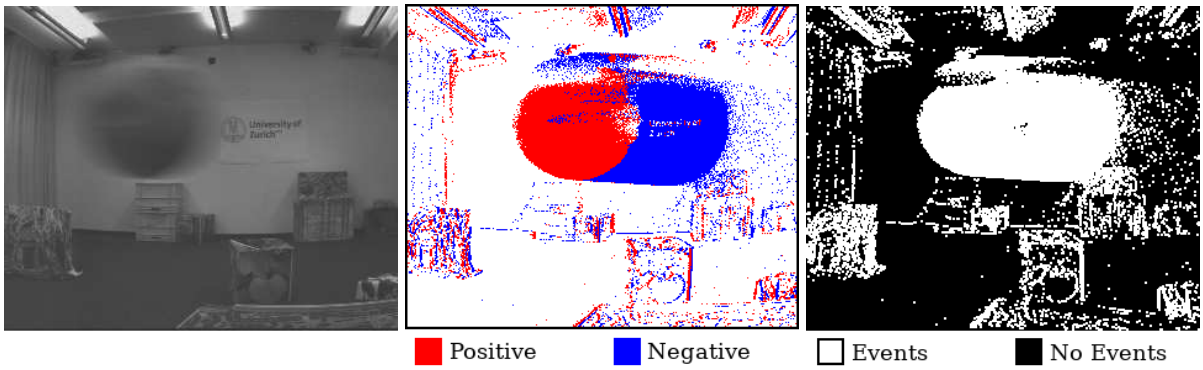
Trading-off detection accuracy for latency is not only a necessity for robotic platforms, but it has been frequently observed also among animals (58) for the execution of several tasks involving



(a) The 3D volume of the events generated within a time window of $10\,\mathrm{ms}$.

(b) The same events, projected into the image plane after ego-motion compensation.

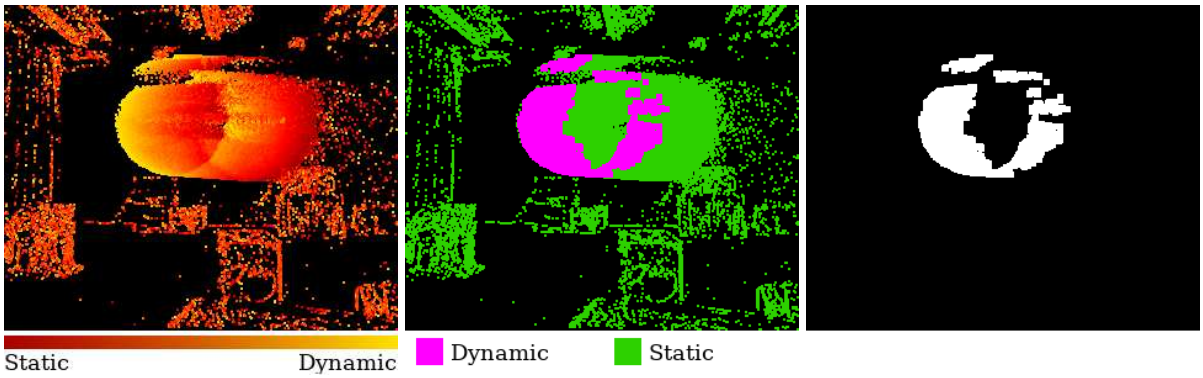
Figure 3: Our algorithm collects all the events that fired during the last $10\,\mathrm{ms}$, here represented in the 3D volume on the left side, and used the Inertial Measurement Unit to compensate for the motion of the camera. The ego-motion compensated events are therefore projected into a common image frame, here shown on the right side, where each pixel contains potentially multiple events. By analyzing the temporal statistics of all the events projected into each pixel, our approach is able to distinguish between pixels belonging to the static part of the scene and to moving objects.



(a) Frame provided by the Insightness SEES1, which shows clearly the motion blur due to the relative motion between the sensor and the moving obstacle..

(b) All the events accumulated in a time window. Red indicates positive events, while negative events are reported in blue.

(c) The same events, motion-compensated using the IMU: if a pixel is colored in white, it means that at least one event has been backprojected there after motion-compensation.



(d) The motion-compensated events, with color code representing the normalized mean timestamp (Eq. (4)): the events belonging to the dynamic part of the scene are represented in yellow.

(e) Mean-timestamp image after threhsolding: green and purple indicate the static and the moving part of the scene, respectively.

(f) Events belonging to moving obstacles. This frame is used to segment out the different dynamic objects in the scene (Sec. Obstacle Segmentation).

Figure 4: A figure summarizing all the steps of our ego-motion compensation algorithm to isolate the events belonging to moving obstacles. Fig. 4a shows a frame captured by the Insightness SEES1 camera. Fig. 4b reports all the events accumulated in the last window, with red and blue indicating the polarity (positive and negative, respectively). Fig. 4c reports the result of the ego-motion compensation, showing in white all the pixels where there has been at least one event in the time window. We compute the normalized mean timestamp of all the events belonging to a given pixel, and the resulting values are shown in Fig. 4d. Based on the normalized mean timestamp, we can disambiguate between the events belonging to the static part of the scene and those belonging to the dynamic objects (Fig. 4e, where green represents static events and purple moving events). Finally, we obtain a frame containing only the events belonging to the dynamic part of the scene, as shown in Fig. 4f.

visual sensing.

We demonstrate the effectiveness of our approach in real experiments with a quadrotor platform. We validate our system with both a monocular setup (for obstacles of known size) and a stereo setup (for obstacles of unknown size), both indoors and outdoors. The entire avoidance framework is capable of running in real-time on a small single-board computer on-board the vehicle, together with the entire software stack necessary to let the robot fly (i.e., state estimation, high-level control, communication with the flight controller). Experimental results show that our framework allows the quadrotor to avoid obstacles moving towards it at relative speeds up to $10\,\mathrm{m/s}$ from a distance of around $3\,\mathrm{m}$. The integration of known techniques for event-based obstacle detection (23) and avoidance (57), adapted in order to make it possible for the entire framework to run in real-time on a small computer as well as to deal with fast-moving obstacles, represents the main contribution of this paper. To the best of our knowledge, this is the first work showing a quadrotor avoiding fast-moving obstacles using only on-board sensing and computing.

Results

Evaluation of the Event-Based Obstacle Detector

In this section, we perform a quantitative evaluation of the performance and effectiveness of our algorithm to detect moving obstacles using event cameras. The first analysis we conduct is about the accuracy of the detections. We collected a large dataset of obstacle detections, including ground-truth data from an Optitrack motion-capture system, in order to characterize the detection error of our algorithm, and in Sec. Accuracy and Success Rate we provide the main results for both the monocular and stereo cases.

In Sec. Computational Cost, we analyze the computational cost of the algorithm, providing some details about how each component contributes to the overall time. Finally, in Sec. Different Types of Obstacles we show that our algorithm can detect different sized and shaped obstacles, while in Sec. Detection of Multiple, Simultaneous Obstacles we discuss the detection of multiple, simultaneous obstacles.

Accuracy and Success Rate

We collected a dataset of more than 250 throws, obtaining around 1200 detections, and compared the output of our event-based detector with ground-truth data from a motion-capture system. For each detection, we computed the norm of the position error, and in Tab. 2, we summarize the results.

We grouped together measurements falling within bins of size $0.5\,\mathrm{m}$, with the first one starting from a distance of $0.2\,\mathrm{m}$ along the camera's optical axis, since the algorithm did not successfully detect the obstacles at closer ranges. In the case of the monocular detector, it was necessary to discard some of the data we collected due to the fact that, at short distances, the obstacle is often only partially visible, and therefore our monocular algorithm fails to correctly estimate

	Monocular				Stereo			
Distance [m]	Mean	Median	Std. Dev.	M.A.D.	Mean	Median	Std. Dev.	M.A.D.
0.2 - 0.5 m	0.08	0.05	0.18	0.09	0.07	0.05	0.07	0.06
0.5 - 1.0 m	0.10	0.05	0.22	0.10	0.10	0.05	0.18	0.10
1.0 - 1.5 m	0.10	0.05	0.20	0.10	0.13	0.07	0.21	0.12

Table 2: A table summarizing the accuracy of our event-based algorithm to detect moving obstacles. We analyzed both the monocular and the stereo setups, and compared the detections with ground-truth data provided by a motion-capture system. For each configuration, we report (expressed in meters) the mean, the median, the standard deviation and the maximum absolute deviation of the norm of the position error, for different ranges of distances.

its distance since it would fit a known size to a partially visible object. This issue becomes less significant as the distance to the camera increases, and after $1\,\mathrm{m}$, it does not significantly impact the detector's performance. On the other hand, as expected, the stereo configuration is more precise at low ranges: the further the distance between the cameras and the object, the higher the uncertainty in the triangulation and, therefore, the larger the error.

Independently of the configuration, however, the data in Tab. 2 show that our algorithm, although not tailored towards accuracy but rather optimized for low latency, provides measurements that are sufficiently accurate to allow a quadrotor to perceive its surroundings and detect moving obstacles effectively. Indeed, the position error is on average smaller than the vehicle size, while the standard deviation is a slightly higher but still reasonable up to $1\,\mathrm{m}$. Among the factors that contribute to the error in estimating the obstacles' position, the low resolution of the camera certainly plays a key role. In Sec. Experiments, we discuss this, as well as other drawbacks of current event cameras.

Another important aspect to consider in our detection algorithm is its success rate. For the entire framework to be effective, not only it has to guarantee low latency, but also it has to guarantee robustness in terms of success rate. In order to assess the robustness of our algorithm, we performed an evaluation with objects of different sizes and shapes. Such objects were thrown through the field of view of the sensors and, using the intrinsic calibration of the camera and ground-truth data from a motion-capture system, we obtained information about when they were supposed to be visible. If the object was in the field of view of the cameras, but the system did not report any detection, we considered it as a failure of the detection algorithm. This allowed to analyze the success rate of our algorithm in detect moving objects, which is reported in Tab. 3. We used objects of size up to $30\,\mathrm{cm}$, and in the table we group together objects belonging to three different categories: smaller than $10\,\mathrm{cm}$, smaller than $20\,\mathrm{cm}$ and up to $30\,\mathrm{cm}$. For each category, we provide data about the success rate when the objects move at different distances from the camera, and group the detections according to such a distance.

As one can notice, our algorithm provide a high success rate in different scenarios, with

Size	Distance [m]		
	$\leq 0.5\,\mathrm{m}$	$\leq 1\,\mathrm{m}$	$\leq 1.5\,\mathrm{m}$
$\leq 0.1\,\mathrm{m}$	92%	90%	88%
$\leq 0.2\,\mathrm{m}$	87%	92%	97%
$\leq 0.3\,\mathrm{m}$	81%	88%	93%

Table 3: A table summarizing the success rate of the event-base detector. Each column reports the success rate for objects moving at a certain distance range from the camera. Each row shows the success rate of detecting objects smaller than a certain size. The results are obtained on a dataset comprising 100 throws of objects belonging to each size.

small and large objects. Given the limited field of view of the cameras, large objects have a lower detection rate at short distances, mostly because of their different appearance in the two cameras used in the stereo setup, which makes it hard for the algorithm to match the individual detections. Similarly, small objects are harder to detect at large distances because of the limited angular resolution of the sensor used for the experiments. Additionally, we did not notice any significant impact of the objects' speed or incoming angle on the success rate.

Computational Cost

To quantify the computational cost of our detection algorithm, we ran an extensive evaluation by throwing objects within the field of view of the event camera, while simultaneously rotating it, and measured the time necessary to process all the events that fired within the last time window of $10\,\mathrm{ms}$. Table 4 shows the results of our evaluation, highlighting how each step of the algorithm, described in details in Sec. Obstacle Detection, contributes to the overall computation time. Our evaluation was performed on an NVIDIA Jetson TX2 board, with the algorithm running exclusively on the CPU (i.e., the GPU available on the same board was not used at all). The numbers reported in Tab. 4 refer to the time required to run the detection algorithm with one camera, however running multiple instances of the same algorithm for multiple cameras (as for example in the stereo case) does not affect the performance in a significant way, as the individual parts can be computed in parallel.

The most expensive part of the algorithm is given by the ego-motion compensation, which on average, requires $1.31\,\mathrm{ms}$ (36.80% of the overall time), with a standard deviation of $0.35\,\mathrm{ms}$. As one can imagine, the time necessary for this step depends on the number of events that need to be processed, and Fig. S4 clearly shows a linear dependence between the two. To understand how many events are typically generated in real-world scenarios during our experiments, we collected some statistics about the number of events that the algorithm needs to process. The data we collected that, on average, both indoors and outdoors, the number of events belonging to a time window of $10\,\mathrm{ms}$ spanned between 2000 and 6000.

Another step that depends on the relative motion between the camera and the scene is the

Step	μ [ms]	σ [ms]	Perc. [%]
Ego-Motion Comp.	1.31	0.35	36.80
Mean Timestamp Thresh.	0.98	0.05	27.52
Morphological Ops.	0.58	0.04	16.29
Clustering	0.69	0.20	19.39
Total	3.56	0.45	100

Table 4: The mean μ and standard deviation σ of the computation time of the obstacle detection algorithm proposed in Sec. Obstacle Detection.

clustering of the events belonging to the dynamic obstacles. This step is necessary to understand how many objects are in the scene and to associate each event with them. Clustering the events usually requires $0.69\,\mathrm{ms}$ (19.39% of the overall time), with a standard deviation of $0.20\,\mathrm{ms}$. The actual processing time to cluster the events depend on the number of pixels where events belonging to dynamic obstacles fired, and Fig. S5 shows how long our clustering algorithm takes as a function of the number of pixels to process.

Finally, thresholding the mean timestamp image and applying some morphological operations to the thresholded image do not depend on the number of events to be processed (as shown by their very low standard deviations), since the entire picture has to be processed, and they require on average $0.98\,\mathrm{ms}$ (27.52%) and $0.58\,\mathrm{ms}$ (16.29% of the overall time), respectively.

It is important to notice that in our evaluation of the algorithm's computational time, we neglected the estimation of the 3D position of the obstacle. This step requires very simple calculations (c.f. Sec. 3D-Position Estimation), which are independent on the number of events generated and on average require times in order of few µs. Therefore, their impact on the overall computational time is negligible.

Different Types of Obstacles

The main reason for us to adopt a stereo configuration for our framework is the necessity to be able to detect moving obstacles independently on their shape and size correctly. Using a single camera, this is not possible as long as the size of the obstacle is not known in advance. Figure S6 shows that the algorithm we propose in this paper to detect moving obstacles using two event cameras is able to detect different kinds of obstacles. In that figure, one can notice how obstacles with completely different geometries can be detected: a small ball, a box, a whiteboard marker, a frisbee, a quadrotor, and a bowling pin. The first column reports a frame grabbed from the SEES1 camera, where the object is often not clearly visible due to motion blur (we manually highlighted the region where the objects are in the frame with a red circle). The remaining columns depict the previously described steps of our detection algorithm, with the same color code used in Fig. 4.

Detection of Multiple, Simultaneous Obstacles

Thanks to the clustering process proposed in Sec. Clustering and the measurements' association step described in Sec. Obstacle Correspondence, our pipeline is able to deal with multiple obstacles moving in the scene simultaneously.

Figure S7 shows an example where the proposed algorithm correctly detects and clusters the events belonging to three different moving obstacles in the scene. In this case, three small-sized balls (manually highlighted by a red circle to facilitate the reader's understanding) are thrown by hand in front of the camera, and the algorithm successfully associates each event to the object they belong to.

The main limitation of our approach is due to the fact that, when two or more obstacles are very close to each other in the frame containing the events belonging to dynamic objects, it is very hard, if not impossible, to disambiguate among them. This is due to the fact that no prior information about the obstacles is used (e.g., shape or, in the stereo case, size), as well as not exploiting any intensity information (i.e., the frames from the on-board camera) in order to tell apart objects that are impossible to segment out using only events.

In our experimental evaluation, this turned out not to be a real issue for the system itself, since as soon as the overlapping obstacles move away from each other, the system is able to detect them promptly and treat them as separate entities.

Experiments

To validate our obstacle avoidance framework, we conducted a large set of experiments in real-world scenarios. The experiments were executed in two different scenarios, one indoors, the other one outdoors. The indoor experiments were conducted within a motion-capture system, in the same setup we used in our previous work (8), and the aim was twofold: (i) collecting ground-truth data in order to verify the effectiveness of the framework in situations where a collision with the obstacle would have happened (which was checked in post-processing thanks to the data from the motion-capture); (ii) validate the overall framework in an easier setup before moving to more complex scenarios. We used the same quadrotor platform we presented in (8) for the indoor experiments, equipped with a monocular setup. Conversely, the outdoor experiments were conducted using a different vehicle, equipped with a stereo setup, as presented in Sec. Experimental Platform. In the remainder of this section, we provide additional details about both the indoor (Sec. Indoor Experiments) and outdoor (Sec. Outdoor Experiments) experiments.

Indoor Experiments

As previously mentioned, the main goal of the indoor experiments is to determine the effectiveness of our framework to avoid dynamic obstacles by determining if a collision was actually prevented by analyzing the data coming from a motion-capture system. The indoor experiments were realized using the same platform described in (8), in the monocular setup. We repeatedly

threw a ball of known size towards the quadrotor, which used the event camera to detect and avoid it. Using the ground-truth measurements coming from the Optitrack motion-capture system, we could intersect the trajectory of the ball with the position where the vehicle was hovering, in order to determine if, without the execution of the escape maneuver, the ball would have hit the vehicle or not. The outcome of this analysis is that our algorithm is capable of preventing actual collisions between a flying robot and dynamic obstacles, at relative speeds up to $10\,\mathrm{m/s}$, as confirmed by the ground-truth data about the trajectory of the object provided by the motion-capture system.

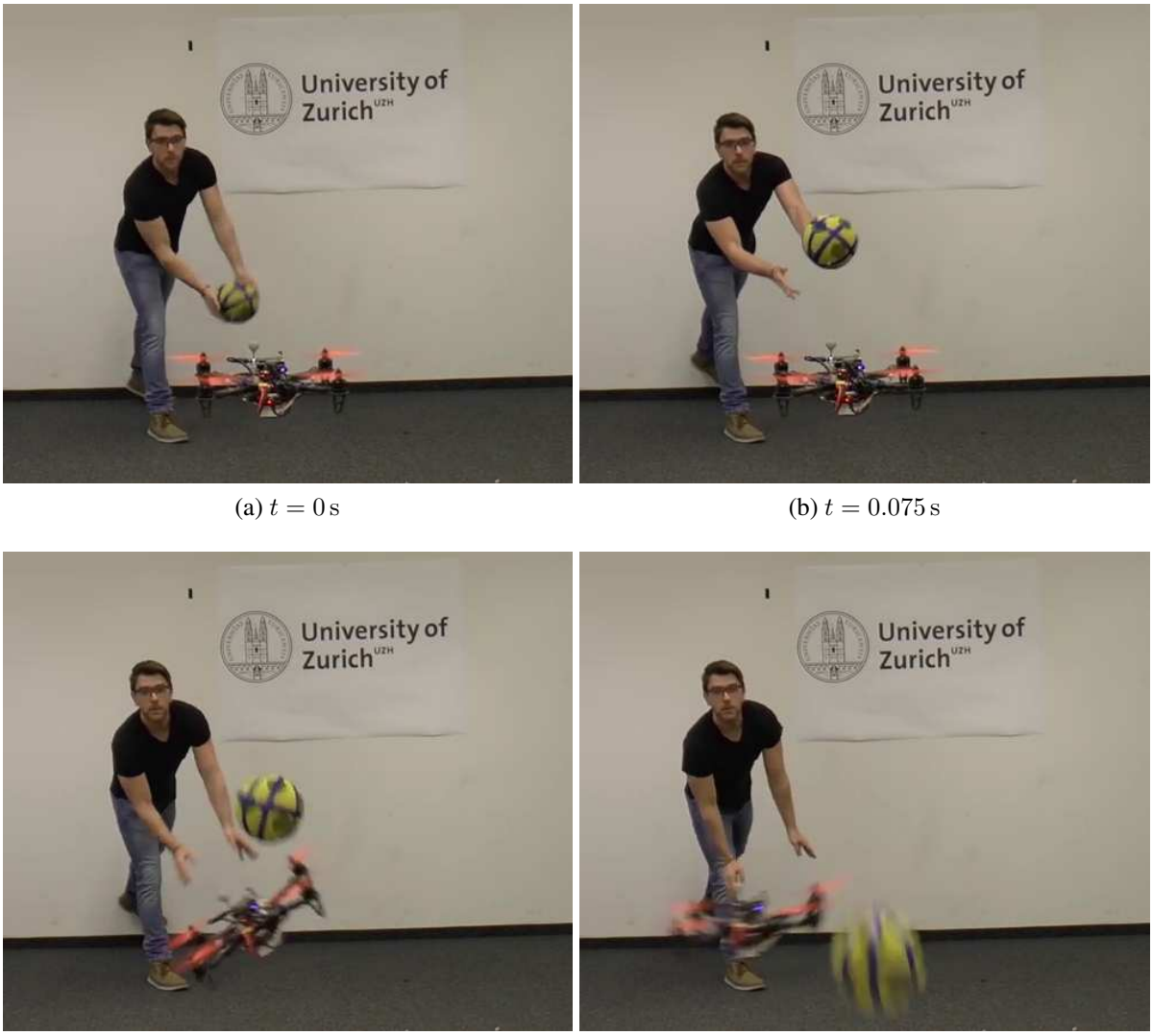
Figure 5 shows one of the indoor experiments, reporting four snapshots recorded with a static camera. The ball takes approximately $0.25\,\mathrm{s}$ to reach the vehicle from the moment it is thrown (Fig. 5a). At that time, as shown in Fig. 5d, the quadrotor already moved to the side to prevent the collision, showing that the algorithm successfully detected the ball and planned an evasive maneuver with very low latency. The experiment reported in Figure 5, as well as other indoor experiments, are shown in the Movie S1.

Outdoor Experiments

After evaluating the performance of our framework in an indoor setup, we performed outdoor experiments using the quadrotor platform described in Sec. Experimental Platform, equipped with two Insightness SEES1 cameras in a stereo setup. We executed two types of experiments, namely in a static scenario, where the vehicle hovers at the desired position, and in a dynamic scenario, where the robot flies towards a target location. In both cases, we threw different kinds of objects towards the quadrotor, which only relied on the two event cameras to detect them and avoid them.

We tested the performance of our algorithm in static scenarios with different types of objects, with multiple obstacles moving towards it at the same time, as well as throwing them consecutively one after the other to benchmark the restiveness out the overall approach. The vehicle successfully manages to detect them and avoid them most of the time, although in some cases, the detection was not successful and led to a collision between the robot and the obstacles. In Sec. Major Failure Causes, Lessons Learnt and Disadvantages of Event Cameras, we discuss the major failure causes of our algorithm; nevertheless, in outdoor experiments, the algorithm successfully detected and avoided the obstacles thrown towards it more than 90% of the time. Movie S2 shows the result of our outdoor experiments in a static scenario.

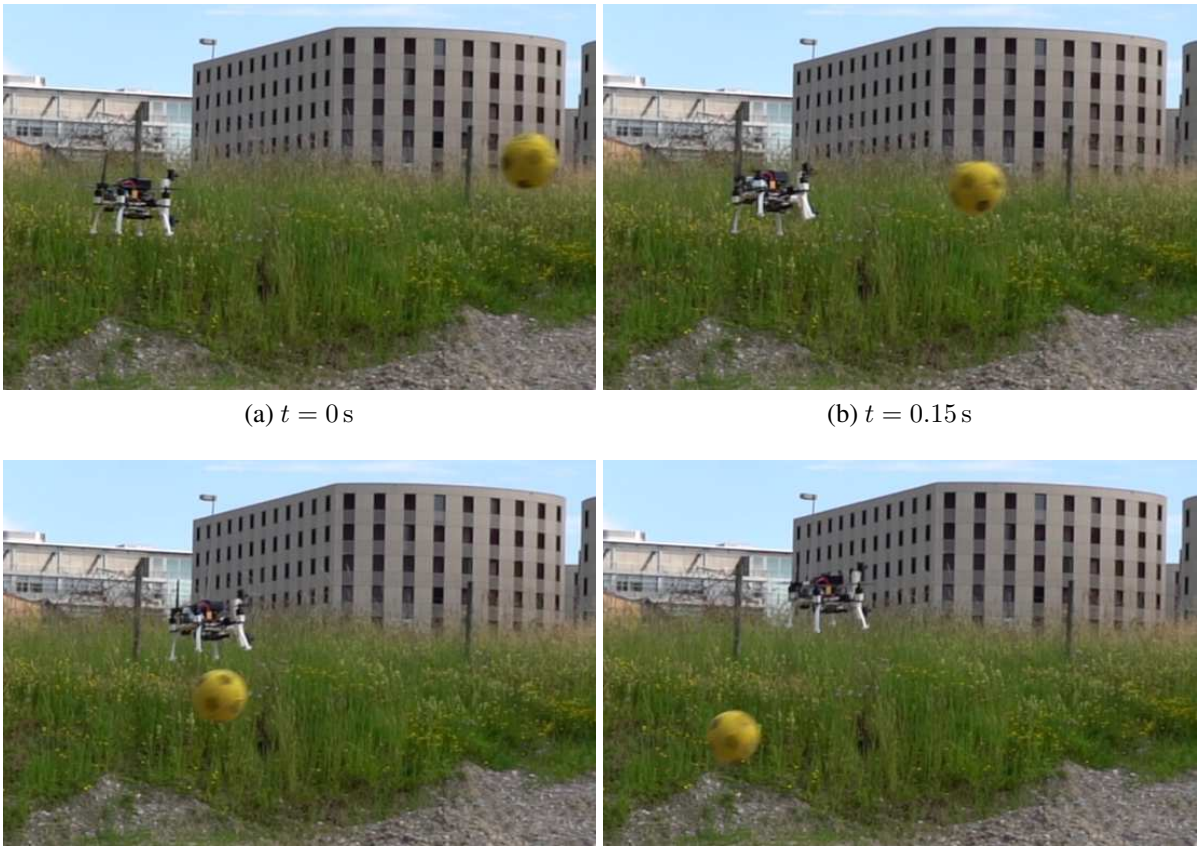
Figure 6 shows four snapshots captured from a sequence recorded in a dynamic scenario. The robot moves towards a target position, from left to right in the pictures, at a linear speed of $1.5\,\mathrm{m/s}$. While reaching its destination, the robot detects the yellow ball thrown towards it (shown on the right side of Fig. 6a). The vehicle decides to execute an evasive maneuver upwards while keeping its travel speed towards the desired position constant. This results in a maneuver that simultaneously allows the vehicle to proceed along with its task and avoid a collision. Additional experiments in a dynamic situation are shown in the Movie S3.



(a) $t = 0\,\mathrm{s}$ (b) $t = 0.075\,\mathrm{s}$

(c) $t = 0.15\,\mathrm{s}$ (d) $t = 0.225\,\mathrm{s}$

Figure 5: A sequence from one of the indoor experiments. A ball is thrown towards the vehicle, equipped with a monocular event camera, which is used to detect and evade the obstacle. The ball is thrown at time $t = 0\,\mathrm{s}$, and reaches the position where the quadrotor is hovering approximately at time $t = 0.225\,\mathrm{s}$. The robot successfully detects the incoming obstacle and moves to the side to avoid it.



(a) $t = 0\,\mathrm{s}$ (b) $t = 0.15\,\mathrm{s}$

(c) $t = 0.30\,\mathrm{s}$ (d) $t = 0.45\,\mathrm{s}$

Figure 6: A sequence from our outdoor experiments. The quadrotor is flying towards a reference goal position when an obstacle is thrown towards it. The obstacle is successfully detected using a stereo pair of event cameras, and is avoided by moving upwards.

Materials and Methods

Obstacle Detection

This section describes how our event-based algorithm to detect moving obstacles works. An additional explanation of the working principle of this algorithm is provided in Movie S4.

Ego-Motion Compensation of the Events

An event camera generates events when intensity changes occur in the image. This can happen because of either moving objects or the ego-motion of the sensor. As we are only interested in avoiding moving objects, the first step is to remove all data generated by the quadrotor's ego-motion.

One way of removing ego-motion from an event stream is described by (23). This approach does, however, utilize an optimization routine to estimate the ego-motion, which is computationally demanding and, therefore, introduces latency in the perception system. In this work, we replace the optimization step with a more simple and computationally efficient ego-motion compensation algorithm. To do this, we use the IMU's angular velocity average over the time window where the events were accumulated in order to estimate the ego-rotation and use this rotation to warp the events in the image. Our approach does not consider the translational motion of the camera but rather assumes that the events are generated mostly by rotational motion. In order to compensate for the translational motion, it would be necessary to estimate the depth of the points generating each event, which would increase the computational complexity too much to be practical. As long as the distance to stationary objects is large enough, our system is not significantly affected by this assumption. Additionally, an analysis of the impact of neglecting the linear component of the robot's ego-motion is provided in Sec. Impact of Using a Simplified Ego-Motion Estimation Algorithm of the supplementary materials. This choice allows our pipeline to be fast enough to guarantee real-time performance but comes at the cost of a potentially higher amount of non-compensated events. To cope with this, we tune the parameters of our algorithm, whose working principle is described below, so that it is able to filter out most of the events generated by the static part of the scene.

The first step of our algorithm requires the collection of a batch of events and IMU data over a specified time δt. In our experiments, we used a time window of length $\delta t = 10\,\mathrm{ms}$, since we realized that this value represents a good compromise between sensitivity and real-time performance. A too short time window renders the entire algorithm too little sensitive, since the events collected do not contain enough information to perform reliable detection. On the other hand, increasing the time window too much leads to a very high number of events to process, making the entire algorithm slower, and does not provide much added value, since the additional events are generated by the past history of the obstacle motion. Next, we average the IMU's angular velocity over δt as $\bar{\boldsymbol{\omega}} = \sum_{\delta t} \boldsymbol{\omega}_t$. We then apply the Rodrigues rotation algorithm to build the rotation matrix from $\bar{\boldsymbol{\omega}}\delta t$ (59). Each event e_i of the batch is then warped in the image plane by $\bar{\boldsymbol{\omega}}(t_i - t_0)$, where t_0 is the time-stamp of the first event of the batch and t_i the

time-stamp of event e_i. This warping is described by a field $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that warps the events' 2D displacement as $\phi(x, y, t - t_0) : (x, y, t) \rightarrow (x', t', t)$. These motion-compensated events are denoted by:

$$C' = \Pi\{\phi(C)\} = \Pi\{\phi(x, y, t - t_0)\} = \{x', y', t_0\} \quad \forall\{x, y, t\} \in C. \tag{1}$$

The original event position (x, y) is part of a discretized image plane in $\mathbb{N}^2$, while (x', y') are part of $\mathbb{R}^2$. From the warped events, we construct the event-count image I, where the pixel value records the total number of events mapped to it by the event trajectory:

$$\xi_{ij} = \{\{x', y', t\} : \{x', y, t_0\} \in C', i = x', j = y'\}. \tag{2}$$

Here $(i, j) \in \mathbb{N}^2$ denotes the integer pixel coordinates of the discretization bin for $(x', y') \in \mathbb{R}^2$. From this we construct the event-count pixel I_{ij} as $I_{ij} = |\xi_{ij}|$, with $|A|$ being the cardinality of the set A. Next, we construct the time-image T, which is also in the discretized plane $\mathbb{N}^2$. Here each pixel contains the average time-stamp of the warped events as:

$$T_{ij} = \frac{1}{I_{ij}} \sum t : t \in \xi_{ij}. \tag{3}$$

In order to determine which pixels belong to a moving object or the background they are each given a score $\rho(i, j) \in [-1, 1]$ for $\{i, j\} \in T$ as:

$$\rho(i, j) = \frac{T(i, j) - \text{mean}(T)}{\delta t}. \tag{4}$$

These scores produce the so called normalized mean time-stamp image ρ. Now, if $\rho(i, j) \geq \tau_{threshold}$, with $\tau_{threshold}$ being a specified threshold, the pixel belongs to a moving object, otherwise to the background.

While the original approach (23) uses a fixed threshold to distinguish between ego-motion generated events and those generated by a moving object, we instead use a linear function that depends on the angular velocity's magnitude, i.e. $\tau_{threshold}(\boldsymbol{\omega}) = a \cdot ||\boldsymbol{\omega}|| + b$. Here a and b are design parameters, where b regulates the threshold while the camera is static and a increases it with an increase of the angular velocity's magnitude. This has the advantage that it is easier to detect moving objects while the quadrotor is static, while still reducing the increased noise generated by faster rotational velocities. After thresholding, it can happen that some events belonging to the static part of the scene are not filtered out, generating some salt and pepper noise that we remove using morphological operations.

It is important to notice that, since our algorithm relies only on the IMU to perform ego-motion compensation, it is less computationally demanding than the approach in (23), but at the same time also more sensitive to false positive detection generated by the ego-motion of the vehicle. To cope with this, we adopted fairly large values for the threshold parameters, in order to avoid false positive. This came at the cost of a less sensitive detection algorithm (i.e.,

it discards more detections that it would with lower thresholds), and therefore we had to find a good compromise between sensitivity and reliability.

Figure 4 shows our algorithm in action. All the events generated in the last time window (Fig. 4b) are motion-compensated using the IMU and, for each pixel, we compute the normalized mean timestamp (Fig. 4d), which is then thresholded (Fig. 4e) to obtain a frame containing only events belonging to moving obstacles (Fig. 4f).

The same algorithm running across different consecutive time windows is shown in Fig. S8. Each column corresponds to a different time, with the first row reporting the frame captured by the on-board camera, the second row showing the events collected in the window, and the third row presenting the same events after the ego-motion compensation and thresholding of the normalized mean timestamp.

Obstacle Segmentation

After performing the ego-motion compensation of the events that fired in the last time window, we obtain a frame containing the location of the events belonging to the dynamic part of the scene (Fig. 4f). It is important to note that at this point, our algorithm already discarded all the static parts of the scene, with a very little computational cost. To do so with a standard camera, one has to receive at least two frames in order to be able to distinguish between static and dynamic objects, and each frame needs to be entirely processed. The output of an event camera, instead, is much more sparse, allowing us only to process the pixels where at least one event fired.

In the remainder of this section, we describe how we use a frame like the one in Fig. 4f in order to cluster together the pixels belonging to the same object.

Clustering

The thresholded image created by the ego-motion compensation described in Section Ego-Motion Compensation of the Events can include multiple moving obstacles, as well as noise. Therefore, the next step is to separate the image points of the individual objects, as well as the noise.

The goal for the system is to be capable of handling an arbitrary number of obstacles, as well as being robust against noise. Additionally, due to the low latency requirement, the clustering has to be performed in the shortest time possible. With these requirements, we evaluated different algorithms in order to decide on the best fitting one for our system.

Our experimental evaluation highlighted that the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm (60) has all the required characteristics for low-latency detection of moving objects. Indeed, it detects clusters without previous knowledge about their shape or their amount. Additionally, it can handle noise by combining it into a separate category. It has an average time complexity of $O(n \log(n))$ and a maximum one of $O(n^2)$, but without the need for an iterative solution, which makes it comparatively fast. Another advantage is that its cost function can be arbitrarily chosen and, therefore, be optimized for our system. Besides the cost function, it only has two design parameters: the minimum number of data points within a

cluster and the maximum cost ϵ for choosing whether a data point belongs to a given cluster. A detailed description of it is found in (60).

Optical Flow

The density of image points, and their distance in the image plane depends on the objects' velocity, distance to the sensor, as well as their overall size. Having only the mean time-stamp information and image position resulting from the ego-motion compensation, as described in Sec. Ego-Motion Compensation of the Events, makes it impossible to effectively cluster the image points of objects with different velocities and distances from the DVS. Therefore, we require additional features. One available possibility is to calculate the image points optical-flow and, therefore, get an estimate of their image-plane velocity. An added advantage is that two objects that generate image-point clusters in close proximity to each other but move in different directions, are easier to distinguish. Ideally one would directly calculate the optical-flow from the event data, but existing algorithms for this either only produce the velocities magnitude or direction, or are extremely computationally expensive while having a low accuracy as evaluated in (61). Instead, we decided to use a conventional optical-flow algorithm on the unthresholded normalized mean time-stamp image produced by the ego-motion compensation. The high temporal resolution of the DVS and high update frequency of our system allows us to assume that the displacement between two frames is small and approximately constant in a region around an image point. Therefore, we use the Lucas-Kanade algorithm (62), which has the advantage that it is less sensitive to noise compared to point-wise methods and by combining the information of several nearby points it is better at handling the ambiguity of the optical-flow equations. To increase the robustness of the optical flow, we apply an averaging filter both to the input images, as well as the resulting velocity field.

Combined Clustering Algorithm

To maximize the accuracy of the clustering, we utilize all the available data information: the image position $\mathbf{p}$, the normalized mean time-stamp value ρ and the, through optical-flow estimated, velocity $\mathbf{v}$. With these quantities, we constructed the DBSCAN's cost function as:

$$C_{i,j}(\mathbf{p}, \mathbf{v}, \rho) = w_p||\mathbf{p}_i - \mathbf{p}_j|| + w_v||\mathbf{v}_i - \mathbf{v}_j|| + w_\rho|\rho_i - \rho_j|. \tag{5}$$

Here $\mathbf{w} = [w_p, w_v, w_\rho]^T$ is a weight vector for the influence of the individual parts.

Even though the DBSCAN algorithm is quite efficient with a maximum data size scaling of $O(n^2)$ the computation time increases with the data size. Especially for fast-moving objects, or ones that move close to the sensor, the density of the generated events and, therefore, the overall data size to be clustered increases. This leads to far greater computation time. To overcome this, we perform a pre-clustering step of the image points using an eight-way connected components clustering algorithm. For this, we assume that two image points that are located directly next to each other in the image plane always belong to the same object. We then calculate the

mean velocity of the image points belonging to the cluster, as well as the mean normalized mean time-stamp and fit a rotated rectangle around the points. The DBSCAN's cost function is adapted to the new features. Instead of using the individual point's velocity and normalized mean time-stamp, we use their corresponding mean values, while the difference in position is substituted by the minimal distance of the corresponding rectangles as:

$$C_{i,j} = w_p \, \mathrm{dist}_{\min}(r_i, r_j) + w_v ||\mathbf{v}_{\text{mean},i} - \mathbf{v}_{\text{mean},j}|| + w_\rho |\rho_{\text{mean},i} - \rho_{\text{mean},j}|. \tag{6}$$

If two corresponding rectangles should overlap, their distance is set to zero. Instead of using rectangles, ellipses could have been used, but finding the minimal distance between two ellipses requires the root calculation of a fourth-order polynomial, requiring an iterative solution, which takes drastically more time. As the connected components algorithm has a time complexity of $O(n)$ and reduces the DBSCAN's data size by orders of magnitude, the overall clustering computation time was decreased on average by a factor of 1000.

3D-Position Estimation

After receiving a set of cluster points, we first fit a rotated rectangle around them to reduce the data dimensionality. From this, we get the four corner points, as well as the center position in the image plane.

For the next step, the estimation of the obstacle's depth towards the image plane, we have to distinguish between the Monocolur and Stereo Case.

Monocular Case. As we are not able to calculate the depth of an image point from a single monocular image, we instead limit our system to objects of known size. With the added size information, we can then estimate the depth of an object in the camera's frame of reference as:

$${}_C\hat{z} = \frac{f \omega_{\text{real}}}{\hat{\omega}}, \tag{7}$$

where f is the focal length, ω_{real} the width of the object and $\hat{\omega}$ the measured side length of the fitted rectangle.

Stereo Case. For the stereo case, we use the disparity between two corresponding clusters of the stereo image pair for the depth estimation. This allows the algorithm to function with objects of unknown size. To determine cluster correspondences, we utilize a matching scheme minimizing the cost:

$$C = w_p \, |x_{c,\text{top}} - x_{c,\text{bottom}}| + w_a \max\left(\frac{A_{\text{top}}}{A_{\text{bottom}}}, \frac{A_{\text{bottom}}}{A_{\text{top}}}\right) + w_n \max\left(\frac{n_{\text{top}}}{n_{\text{bottom}}}, \frac{n_{\text{bottom}}}{n_{\text{top}}}\right) - 2, \tag{8}$$

with $\mathbf{w} = (w_p, w_a, w_n)$ being weights, x_c the cluster's center's position in the image plane, A the fitted rectangle's area and n the number of cluster points. Next, we use the cluster's disparity to calculate the depth as described in (63). To increase the robustness, we use the cluster's centers to estimate the depth instead of directly projecting the corner points into 3D space. Having

estimated the obstacle's depth, we approximate its size using the rearranged formulation as in the monocular case as:

$$\omega_{\text{est}} = \frac{{}_C\hat{z}\hat{\omega}}{f}. \tag{9}$$

Image to World Projection

With the obtained obstacle's depth and size, we now project the cluster's corner and center points into 3D space using the perspective projection model in homogeneous coordinates:

$$\lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{K}_C\mathbf{X}_i, \tag{10}$$

with $\mathbf{K}$ being the intrinsic camera matrix, λ_i the scale factor, ${}_C\mathbf{X}_i$ the Cartesian coordinates of each point of the cluster in the camera frame, and u_i and v_i the pixel coordinates of their projection in the image. The points ${}_C\mathbf{X}_i$ are then transformed into the world's frame of reference by applying:

$$\begin{bmatrix} {}_W\mathbf{X}_i \\ 1 \end{bmatrix} = \mathbf{T}_{WB}\mathbf{T}_{BC} \begin{bmatrix} {}_C\mathbf{X}_i \\ 1 \end{bmatrix}, \tag{11}$$

where $\mathbf{T}_{WB}$ and $\mathbf{T}_{BC}$ are transformation matrices representing the pose (rotation and translation) of the body frame with respect to the world frame, and of the camera with respect to the body frame, respectively. Here the center-point's depth is both increased and decreased by the obstacle's estimated size as:

$${}_C\hat{z}_{c,\pm} = {}_C\hat{z} \pm \omega_{\text{est}}. \tag{12}$$

This gives us a total of six points ${}_W\mathbf{X}_{1:6}$ representing the obstacle.

Obstacle Correspondence

In order to estimate an obstacle's velocity, we first have to determine if a newly detected obstacle corresponds to a previous one and, if this is the case, to which. This is done by finding the best match between the new obstacle's center and the predicted position of the saved obstacles' centers. This is done by finding the closest position match within a sphere around the newly detected obstacle.

Obstacle Velocity Estimation

Once the 3D position of an obstacle has been estimated, our algorithm requires some further processing in order to provide valuable information to the planning stage, for a twofold reason: (i) the event-based detections are sometimes noisy, especially at large distances; (ii) it is necessary to estimate the obstacle's velocity, which is used to determine the avoidance direction, as well as a scaling factor for the repulsive potential field (Sec. Obstacle Avoidance). To do so, we

use a Kalman filter (64), with the obstacle's position estimate as input for the measurement update. This introduces some time lag (typically below $0.3\,\mathrm{ms}$) as the Kalman filter behaves as a low-pass filter, but the increased accuracy is, in this case, preferable. For this we assume a constant velocity model having as state the obstacle's position and velocity:

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \dot{\mathbf{x}}_{k-1}\,\Delta t \tag{13}$$
$$\dot{\mathbf{x}}_k = \dot{\mathbf{x}}_{k-1} \tag{14}$$
$$\Delta t = t_k - t_{k-1}. \tag{15}$$

With this we can formulate the linear motion model as:

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \dot{\mathbf{x}} \end{bmatrix} \tag{16}$$
$$\hat{\mathbf{x}}_k = \mathbf{A}_k\,\hat{\mathbf{x}}_{k-1} + \mathbf{v}_k \tag{17}$$
$$\mathbf{z}_k = \mathbf{H}\,\hat{\mathbf{x}}_k + \mathbf{w}_k \tag{18}$$
$$\mathbf{A}_k = \begin{bmatrix} \mathbf{I}_{3x3} & \Delta t \cdot \mathbf{I}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{I}_{3x3} \end{bmatrix} \tag{19}$$
$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix}, \tag{20}$$

where $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the normally distributed process noise and $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the normally distributed measurement noise. Using this we now construct the Kalman Filter as follows (64):

$$\hat{\mathbf{x}}_{p,k} = \mathbf{A}_k\,\hat{\mathbf{x}}_{m,k-1} \tag{21}$$
$$\mathbf{P}_{p,k} = \mathbf{A}_k\,\mathbf{P}_{m,k-1}\,\mathbf{A}_k^T + \mathbf{Q} \tag{22}$$
$$\mathbf{K}_k = \mathbf{P}_{p,k}\,\mathbf{H}^T\,(\mathbf{H}\,\mathbf{P}_{p,k}\,\mathbf{H}^T + \mathbf{R})^{-1} \tag{23}$$
$$\hat{\mathbf{x}}_{m,k} = \hat{\mathbf{x}}_{p,k} + \mathbf{K}_k\,(\mathbf{z}_k - \mathbf{H}\,\hat{\mathbf{x}}_{p,k}) \tag{24}$$
$$\mathbf{P}_{m,k} = (\mathbf{I} - \mathbf{K}_k\,\mathbf{H})\mathbf{P}_{p,k}(\mathbf{I} - \mathbf{K}_k\,\mathbf{H})^T + \mathbf{K}_k\,\mathbf{R}\,\mathbf{K}_k^T. \tag{25}$$

This has the added advantage that we receive a filtered estimate of the obstacle's velocity without any further computations.

Obstacle Avoidance

The primary objective of our avoidance framework is to guarantee low latency between sensing and actuation. The low latency on the perception side is guaranteed by the previously described event-based obstacle detection pipeline. For the overall system to be effective, however, it is necessary to reduce the latency of the decision-making system responsible for driving the robot away from the detected obstacles. Based on this consideration, it is intuitive to understand that any optimization-based avoidance technique is not suited for our purpose since numerical optimization would introduce latency due to the non-negligible computation times. Rapid

methods to compute motion primitives for aerial vehicles exist in the literature (65). However, they present a number of drawbacks. First, it is necessary to sample both space and time to find a safe position for the robot and a suitable duration of the trajectory. Additionally, continuity in the control inputs is not always guaranteed. Finally, including this kind of methods within existing motion generation frameworks is not always trivial due to multiple reasons: it is necessary to continuously switch between the main navigation algorithm, driving the robot towards its goal, and the avoidance algorithm, steering it away from obstacles; it is not always trivial to obtain a behavior that allows the robot to keep executing its mission (e.g., reach its goal) while simultaneously avoiding moving obstacles.

The artificial potential field method is a natural and simple solution to all the aforementioned issues. Given a closed-form expression of the attractive and repulsive fields, it is particularly simple to compute their gradients within negligible computation time in order to generate the resulting force responsible for letting the robot move. Considering an obstacle as the source of a repulsive field also allows us to not require any sampling in space and time since the resulting potential decides in which direction the robot should move at each moment in time. Finally, the resulting potential can be used at different levels of abstraction in order to integrate the command derived from its gradient into existing motion generation algorithms, for example, as velocity or acceleration commands.

Using potential fields for pathfinding and obstacle avoidance has been extensively researched. This approach is, however, mostly used in static scenarios, whereas our system is thought for dynamic obstacles. The typical approach is to build a discretized map, where each element represents the potential combined from the attractive and repulsive parts. This map building approach is feasible in 2D, but its size and the required computational power to build and analyze it drastically increase when doing so in 3D, as it increases from $O(n^2)$ to $O(n^3)$. Instead of building a map, we represent the obstacles as a struct of features, resulting in a sparse and minimal data representation. The obstacles are represented as ellipsoids, with a potential that is decaying over time. We use the estimated obstacles' position and velocity to calculate their repulsive forces at each time step. Additionally, given a reference target position, we compute the attractive force towards it. With the combined force, we then produce a velocity command which is sent to the controller. Additionally, the system's behavior, when no obstacles are present, is similar to the one generated by a high-level trajectory planner driving the robot towards the desired goal location.

Obstacles Representation

We chose to represent the obstacles as ellipsoids, as they are a good representation of the expected Gaussian error of both the position and size. Additionally, they allow us to generate a continuous repulsive force when an obstacle is detected. Using the six coordinate points ${}_W\hat{X}_{1:6}$ in Eq. 12, we fit a minimal volume ellipsoid around them using the approach described in (66) and illustrated in Fig. S9.

Repulsive Potential Field

Each obstacle produces a potential field $U_{r,i}$, from which we get the repulsive force $\mathbf{F}_{r,i}$ by calculating its gradient as $\mathbf{F}_{r,i} = -\nabla U_{r,i}$. One way of formulating the potential field was proposed by (67), which in turn is a modification of the original artificial potential field definition by (57), as:

$$U_{r,i}(\eta_i) = \begin{cases} k_{r,i}\left(\frac{\eta_0 - \eta_i}{\eta_0}\right)^{\gamma}, & \text{if } 0 \leq \eta_i \leq \eta_0 \\ \mathbf{0}, & \text{if } \eta_i > \eta_0 \end{cases}, \tag{26}$$

with a resulting force:

$$\mathbf{F}_{r,i} = -\nabla U_{r,i} = \begin{cases} \frac{k_{r,i}\gamma}{\eta_0}\left(\frac{\eta_0 - \eta_i}{\eta_0}\right)^{\gamma-1}\nabla\eta_i, & \text{if } 0 \leq \eta_i \leq \eta_0 \\ \mathbf{0}, & \text{if } \eta_i > \eta_0 \end{cases}, \tag{27}$$

where k_r, γ and η_0 are design parameters and η_i is the distance to the obstacle i. This kind of field does, however, produce a gradient whose magnitude increases slowly as the distance to the obstacle decreases, as shown in Figure S10a. This has the effect that the repulsive force acting on the quadrotor only reaches significant values when the obstacle is close, or a high repulsive gain k_r has to be chosen, which might lead to unstable, aggressive behavior.

Therefore, we propose a new formulation of the repulsive force as:

$$||\mathbf{F}_{r,i}|| = \begin{cases} k_{r,i}\left(1 - \frac{1 - e^{\gamma\eta_i}}{1 - e^{\gamma\eta_0}}\right), & \text{if } 0 \leq \eta_i \leq \eta_0 \\ 0, & \text{if } \eta_i > \eta_0 \end{cases}, \tag{28}$$

as shown in Figure S10b. Here η_i is the minimal distance to the ellipsoid's surface of obstacle i. Through this formulation, the force's magnitude is limited to a specified value k_r and increases much faster. This is desirable when evading fast-moving obstacles, as compared to static ones, for which the fields described in other works were developed, as the quadrotor's dynamics require it to start evading before an obstacle comes too close, as discussed in (38).

Conventionally, the gradient of the distance towards the obstacle $\nabla\eta_i$ is responsible for the direction of the repulsive force $\mathbf{F}_{r,i}$. It points in the direction of the steepest descent of the obstacle's distance, which is the opposite direction between the quadrotor's center and the closest point on the obstacle's ellipsoid's surface. This means that an obstacle pushes the quadrotor away from it. We do, however, want to apply a different avoidance strategy. Instead, we use the obstacle's predicted velocity $\dot{\mathbf{x}}_i$ and the distance's gradient $\nabla\eta_i$ and calculate the normalized cross product as:

$$\boldsymbol{\theta}_i = \frac{\nabla\eta_i \times \dot{\mathbf{x}}_i}{||\nabla\eta_i \times \dot{\mathbf{x}}_i||}. \tag{29}$$

Next, we project this vector into the plane orthogonal to the quadrotor's heading $\boldsymbol{\theta}_{\text{quadrotor}}$ as:

$$\boldsymbol{\theta}_{i,n} = \boldsymbol{\theta}_i - < \boldsymbol{\theta}_i, \boldsymbol{\theta}_{\text{quadrotor}} > \boldsymbol{\theta}_{\text{quadrotor}}. \tag{30}$$

With the new avoidance direction $\boldsymbol{\theta}_{i,n}$ the repulsive force $\mathbf{F}_{r,i}$ becomes:

$$\mathbf{F}_{r,i} = -\nabla U_{r,i} = \begin{cases} k_{r,i}\left(1 - \dfrac{1 - e^{\gamma\eta_i}}{1 - e^{\gamma\eta_0}}\right)\boldsymbol{\theta}_{i,n}, & \text{if } 0 \leq \eta_i \leq \eta_0 \\ \mathbf{0}, & \text{if } \eta_i > \eta_0 \end{cases}. \tag{31}$$

This formulation of the potential field yields to a behaviour such that, if the quadrotor is moving towards the goal location, it flies around any detected obstacle if the goal position is behind it, while if it is in hover conditions it moves in a direction orthogonal to the obstacle's velocity. Finally, we include the magnitude of the obstacle's estimated velocity $||\dot{\mathbf{x}}_i||$, into the repulsive force $\mathbf{F}_{r,i}$ as:

$$\mathbf{F}_{r,i} = -\nabla U_{r,i} = \begin{cases} ||\dot{\mathbf{x}}_i|| k_{r,i}\left(1 - \dfrac{1 - e^{\gamma\eta_i}}{1 - e^{\gamma\eta_0}}\right)\boldsymbol{\theta}_{i,n}, & \text{if } 0 \leq \eta_i \leq \eta_0 \\ \mathbf{0} & \text{if } \eta_i > \eta_0 \end{cases}. \tag{32}$$

By doing so, faster obstacles produce a larger repulsive force and the quadrotor will therefore perform a more aggressive avoidance maneuver. This is desirable since the faster an obstacle, the lower the avoidance time, which therefore implies the necessity for a quick evasive maneuver.

Additionally, we ensure that the z-component of the repulsive force is always positive, namely $F_{r,i,z} = |F_{r,i,z}|$, as quadrotors with sufficiently large *thrust-to-weight* ratios are typically capable of producing larger accelerations upwards than downwards.

The repulsive constant $k_{r,i}$ is in our case dynamic and decays with time as:

$$k_{r,i}(t) = k_{r,0}\, e^{-\lambda_{decay}(t - t_{detection,i})}, \tag{33}$$

where $k_{r,0}$ is the initial repulsive constant, λ_{decay} a factor regulating the decay rate, t the current time and $t_{detection,i}$ the last time the specific obstacle was detected. Through this decay, obstacles are kept in case of a temporary occlusion or when they leave the camera's field of view. Their effect on the quadrotor, however, decreases as the time to their last detection increases. If $k_{r,i}$ falls below a given threshold $k_{r,\tau}$, the obstacle is removed.

Finally, the parameter η_i represents the minimal distance between the quadrotor's center to the obstacle's ellipsoid's surface minus the quadrotor's radius. The computation of the minimal distance between a point and an ellipsoid's surface is described in (68).

The total repulsive force is then the sum over all individual obstacles as:

$$\mathbf{F}_{r,total} = \sum_i \mathbf{F}_{r,i}. \tag{34}$$

Attractive Potential Field

The goal of the attractive potential field is to allow the vehicle to reach the desired target position and hover there until the user provides a new reference. In this work, we provide a simple

formulation for the attractive potential that assumes that no static obstacles are present in the scene, i.e., the straight-line path between the robot and the obstacle is collision-free. However, one can easily replace this component of our avoidance scheme with more sophisticated methods to generate commands that drive the vehicle towards its goal. These can be based, for example, on potential field-based techniques dealing with static obstacles and local minima, which is out of the scope of this work, or completely different methods able to generate velocity or acceleration commands (for example (69)).

For the attractive potential, we want the system to produce the same velocity towards a goal as a high-level planner would produce if no obstacle is present, but also produce stable dynamics close to the goal. Therefore, we chose the hybrid approach of a conical and polynomial potential field (70) as:

$$U_a = \begin{cases} \frac{k_a}{(\gamma_a+1)e_0^{\gamma_a}}||\mathbf{e}||^{\gamma_a+1}, & \text{if } ||\mathbf{e}|| < e_0 \\ k_a||\mathbf{e}||, & \text{if } ||\mathbf{e}|| \geq e_0 \end{cases}. \tag{35}$$

This function is differentiable at e_0, i.e. the crossover distance between the two different potential fields, with $\mathbf{e}$ being the error between the goal's and quadrotor's positions, k_a the attractive constant and γ_a a design parameter. By taking its gradient we get the attractive force as:

$$\mathbf{F}_a = -\nabla U_a = \begin{cases} k_a \frac{\mathbf{e}}{||\mathbf{e}||}\left(\frac{||\mathbf{e}||}{e_0}\right)^{\gamma_a}, & \text{if } ||\mathbf{e}|| < e_0 \\ k_a \frac{\mathbf{e}}{||\mathbf{e}||}, & \text{if } ||\mathbf{e}|| \geq e_0 \end{cases}, \tag{36}$$

which is continuous in $\mathbf{e}$. The constant k_a regulates the output velocity $\dot{\mathbf{x}}$, see Section Output Velocity, and by setting it to $k_a = ||\mathbf{v}_{des}||$ the quadrotor's velocity's magnitude is $||\dot{\mathbf{x}}|| = ||\mathbf{v}_{des}||$, while $||\mathbf{e}|| \geq e_0$ and no obstacles are present.

If we would instead solely rely on the conical potential field, the quadrotor would start to oscillate around its goal position, as the resulting force's magnitude would be k_a, regardless of the error. The attractive force's magnitude is shown in Figure S11. If $\gamma_a = 0$ then $||\mathbf{F}_a||$ is identical to that of the conical part, producing a constant magnitude of the attractive force, while for $\gamma_a = 1$ the magnitude goes linearly to 0. With increasing γ_a the magnitude drops faster with an increasingly large area around $||\mathbf{e}|| = 0$, where it is close to 0.

Output Velocity

The velocity is the output of our system and is given to the controller to derive the required total thrust and body-rates. From the total repulsive force $\mathbf{F}_{r,total}$ and attractive force $\mathbf{F}_a$ we get the total virtual force acting on the quadrotor as $\mathbf{F}_{total} = \mathbf{F}_{r,total} + \mathbf{F}_a$. With this force, we now have three possible design choices to calculate the quadrotor's desired velocity $\dot{\mathbf{x}}$:

$$\ddot{\mathbf{x}} = \frac{\mathbf{F}_{total}}{m} \tag{37}$$

$$\ddot{\mathbf{x}} = \mathbf{F}_{total} \tag{38}$$

$$\dot{\mathbf{x}} = \mathbf{F}_{total}, \tag{39}$$

where m denotes the quadrotor's mass.

Both (37) and (38) produce a first order dynamic, while (39) directly produces the velocity output. Introducing further dynamics into the system results in additional delays, which is undesirable since we want our system to be as responsive as possible. We, therefore, chose (39) as it produces the fastest response.

Experimental Platform

Hardware

To validate our approach with real-world experiments, we designed a custom quadrotor platform. The main frame is a 6" Lumenier QAV-RXL, and, at the end of each arm, we mounted a Cobra CM2208-2000 brushless motor equipped with 6", three-bladed propeller. The vehicle is equipped with two on-board computers: (i) a Qualcomm Snapdragon Flight, used for monocular, vision-based state estimation using the provided Machine Vision SDK; (ii) an NVIDIA Jetson TX2, accompanied by an AUVIDEA J90 carrier board, running all the rest of our software stack. In this regard, the output of our framework is a low-level control command comprising the desired collective thrust and angular rates the vehicle should achieve in order to fly. These commands are sent to a Lumenier F4 AIO Flight Controller, which then produces single-rotor commands that are fed to DYS Aria 35a motor controllers.

The quadcopter is equipped with two front-facing Insighteness SEES1 cameras, in a vertical stereo setup, connected via USB to the Jetson TX2. The SEES1 sensor provides both frame and events, and has a QVGA resolution ($320 \times 240\,\mathrm{pxl}$). In order to have a sufficiently high angular resolution, each camera has a lens providing a horizontal field of view of approximately 80°. Such a small field of view is particularly low for tasks such as obstacle avoidance, where a large field of view is preferable to increase the area that the robot can sense. The choice of adopting a vertical stereo setup rather than a more common horizontal setup was driven by the necessity of maximizing the overlap between the field of view of the two cameras while guaranteeing a sufficiently large baseline (in our case, $15\,\mathrm{cm}$).

In addition to the previous sensing suite, we mounted a Teraranger EVO 60m distance sensor looking downwards. The goal of this additional sensor is to constantly monitor the height of the vehicle in order to detect whether there is any drift in the state estimate provided by the Visual-Inertial Odometry (VIO) pipeline running on the Snapdragon Flight. Whenever we detect a discrepancy beyond a manually defined threshold, the quadrotor automatically executes an emergency landing maneuver.

Software

We developed the software stack running on our quadrotor in C++ using ROS for communication among different modules. To reduce latency, we implemented the obstacle detection and avoidance algorithms within the same ROS module, so that no message exchange is necessary between the camera drivers and the code responsible for detecting moving obstacles, as well as

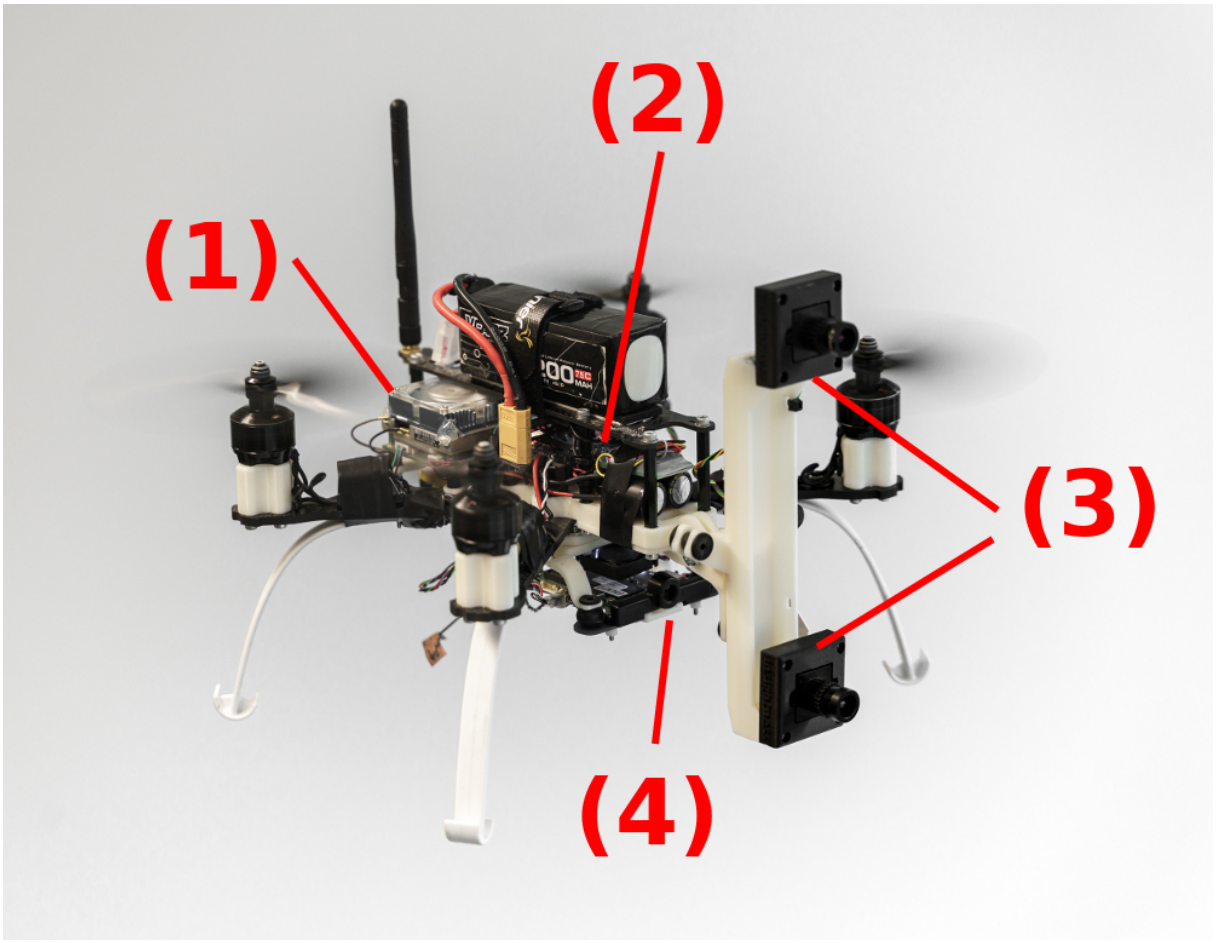


Figure 7: The quadrotor platform we used in our outdoor experiments. The following components are highlighted in the picture: (1) the Nvidia Jetson TX2, running the obstacle detection and avoidance algorithm, as well as the high-level controller; (2) the Lumenier F4 AIO Flight Controller; (3) the two Insightness SEES1 cameras, in a vertical stereo setup; (4) the Qualcomm Snapdragon Flight board, used for state estimation.

between the latter and the planning stage. The output of this module is a velocity command, which is then fed to the position controller proposed in (71) and available as opensource [1]. The low-level controller, responsible for tracking desired body rates and collective thrust, is the default one provided by the Lumenier F4 AIO Flight Controller, which then communicates with the ESCs to generate the single rotor thrusts.

In our outdoor experiments, the state of the vehicle is estimated using the Visual-Inertial Odometry pipeline provided by the Qualcomm Machine Vision SDK [2], which however only provides new estimates at camera rate (up to $30\,\mathrm{Hz}$. This is not sufficient to control our vehicle with low latency and would represent a bottleneck in the entire pipeline. In order to obtain a higher-rate state estimate, we feed the output of the VIO into an Extended Kalman Filter (72), together with IMU measurements, to obtain information about the position, orientation and velocity of the vehicle at $250\,\mathrm{Hz}$.

Major Failure Causes, Lessons Learnt and Disadvantages of Event Cameras

As we have previously shown, event cameras allow fast, low-latency detection of moving obstacles. We discussed in Sec. Event Cameras the advantages of these novel bio-inspired neuromorphic sensors against standard cameras. However, as of today, they are mostly a research-oriented sensor, and thus still require a significant engineering effort in order to solve the main issues they present.

One of the problems with current event cameras is their weight. Most of the event cameras available nowadays are larger and heavier than state-of-the-art standard cameras for robotic applications, which are typically below $50\,\mathrm{g}$. The Insighteness SEES1 is, to the best of our knowledge, the smallest event camera that also provides frames (which is particularly convenient to easily calibrate the intrinsic and extrinsic parameters of the sensor) and can be easily mounted on a quadrotor (its size is $3.5 \times 3.5\,\mathrm{cm}$, and it weighs $15\,\mathrm{g}$). However, its resolution ($320 \times 240\,\mathrm{pxl}$, QVGA) is particularly low compared to standard cameras. This imposes the necessity to find the right trade-off between the field of view and the angular resolution: the larger the first, the smallest the second, which reduces the sensing range at which it is possible to detect objects reliably (8). A small field of view, however, has a negative impact on the detection of obstacles entering the sensing range of the vehicle from the side, as for example in our outdoor dynamic experiments: the larger the field of view, the earlier the vehicle can detect and avoid obstacles moving towards it from the sides.

Another problem characterizing these novel sensors is their noise characteristics. Indeed, these sensors show higher noise than standard cameras, which often has a negative impact on the performance of event-based vision algorithms. In our approach, for example, in order to obtain reliable detections and to eliminate false positives caused by the sensor noise, we had to

[1] http://rpg.ifi.uzh.ch/rpg_quadrotor_control.html

[2] https://developer.qualcomm.com/software/machine-vision-sdk

significantly increase the threshold used to separate events generated by the static part of the scene from those caused by moving objects. This resulted in an obstacle detection algorithm less reactive to small relative motion, especially at large distances. For this reason, we discard all the detections reporting distances between the camera and the obstacle beyond $1.5\,\mathrm{m}$.

The aforementioned reasons represent the main failure causes of our approach. In most of the cases, when our quadrotor was not able to avoid an object thrown towards it, this was due to the fact that it was detected too late, either because it entered the field of view of the camera at a distance that was too short (and therefore the vehicle could not complete the evasive maneuver in time), or because the motion of the obstacle did not generate sufficient events to allow our algorithm to detect it.

Conclusions

We presented a framework to let a quadrotor dodge fast-moving obstacles using only onboard sensing and computing. Different from state of the art, our approach relies on event cameras, novel neuromorphic sensors with reaction times of microseconds. Each pixel of an event camera reacts to changes in intensity, making this sensor a perfect fit for detecting and avoiding dynamic obstacles. Event cameras can overcome the physical limitations of standard cameras in terms of latency, but require novel algorithms to process the asynchronous stream of events they generate.

We investigated the exploitation of the temporal statistics of the event stream in order to tell apart the dynamic part of a scene, showing that it is possible to detect moving objects with a perception latency of $3.5\,\mathrm{ms}$. We showed that our algorithm is capable of accurately and reliably detecting multiple simultaneous objects with different shapes and sizes. We combined our event-based detection algorithm with a fast strategy to generate commands that allow the vehicle to dodge incoming objects. We validated our approach with extensive experiments on a real quadrotor platform, both indoors and outdoors, demonstrating the effectiveness of the method at relative speeds up to $10\,\mathrm{m/s}$.

List of Supplementary Materials

The supplementary PDF file includes:

Figure S1. Monodimensional example to explain the working principle of event-based detection of moving obstacles.
Figure S2. Time statistics of the events belonging to static and dynamic regions.
Figure S3. Ego-motion compensation computation time as function of the number of events.
Figure S4. Clustering computation time as function of the pixels count.
Figure S5. Detection of objects having different sizes and shapes.
Figure S6. Detection of multiple objects simultaneously.
Figure S7. Sequence of detection.
Figure S8. Obstacle ellipsoid.
Figure S9. Repulsive potential.
Figure S10. Attractive potential.

Other Supplementary Material for this manuscript include:

Movie S1 (.mp4 format). Indoor experiments.
Movie S2 (.mp4 format). Outdoor static experiments.
Movie S3 (.mp4 format). Outdoor dynamic experiments.
Movie S4 (.mp4 format). Explanation of the working principle of the event-based detection algorithm.

The materials can be found here: `http://rpg.ifi.uzh.ch/event_based_avoidance`

References

1. "Aerial imaging market size, share and industry analysis by camera orientation (oblique, vertical), platform (fixed-wing aircraft, helicopter, uav/drones), end-use industry (government, energy sector, defense, forestry and agriculture, real estate, civil engineering, insurance) and regional forecast, 2018-2025," *Fortune Business Insights*, 2019. [Online]. Available: http://www.fortunebusinessinsights.com/industry-reports/aerial-imaging-market-100069

2. Markets and Markets, "Autonomous last mile delivery market worth $91.5 billion by 2030," *Bloomberg*, 2019. [Online]. Available: http://www.bloomberg.com/press-releases/2019-07-15/autonomous-last-mile-delivery-market-worth-91-5-billion-by-2030-exclusive-report-by-marketsandmarkets

3. Reports and Data, "Urban air mobility market to reach usd 7.9 billion by 2030," *Globe NewsWire*, 2019. [Online]. Available: http://www.globenewswire.com/news-release/2019/03/18/1756495/0/en/Urban-Air-Mobility-Market-To-Reach-USD-7-9-Billion-By-2030-Reports-And-Data.html

4. G. McNeal, "Video shows kings fans knocking drone out of sky, did it belong to lapd?" *Forbes*, 2014. [Online]. Available: https://www.forbes.com/sites/gregorymcneal/2014/06/14/video-shows-kings-fans-knocking-drone-out-of-sky-did-it-belong-to-lapd/#4377a6584284

5. T. Powell, "Bizarre moment argentine football fan takes down drone with well-aimed toilet roll as it films crowd," *Evening Standard*, 2017. [Online]. Available: http://www.standard.co.uk/news/world/bizarre-moment-argentine-football-fan-takes-down-drone-with-wellaimed-toilet-roll-as-it-films-crowd-a359html

6. M. O. Reporter, "When eagles attack! drone camera mistaken for rival," *Daily Mail*, 2016. [Online]. Available: http://www.dailymail.co.uk/video/news/video-1154408/Golden-Eagle-attacks-drone-camera-mistaking-rival.html

7. A. Domanico, "Hawk attacks drone in a battle of claw versus machine," *CNet*, 2014. [Online]. Available: http://www.cnet.com/news/this-hawk-has-no-love-for-your-drone/

8. D. Falanga, S. Kim, and D. Scaramuzza, "How fast is too fast? the role of perception latency in high-speed sense and avoid," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1884–1891, Apr. 2019.

9. O. Esrafilian and H. D. Taghirad, "Autonomous flight and obstacle avoidance of a quadrotor by monocular slam," in *International Conference on Robotics and Mechatronics (ICROM)*, Oct 2016, pp. 240–245.

10. H. Alvarez, L. M. Paz, and D. Cremers, *Collision Avoidance for Quadrotors with a Monocular Camera*, 2016, pp. 195–209.

11. Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *J. Field Robot.*, vol. 35, no. 1, pp. 23–51, 2018.

12. H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for mav flight," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2015, pp. 50–56.

13. M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2015, pp. 1872–1878.

14. K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar, "Fast, autonomous flight in gps-denied and cluttered environments," *J. Field Robot.*, vol. 35, no. 1, pp. 101–120, 2018.

15. A. J. Barry, P. R. Florence, and R. Tedrake, "High-speed autonomous obstacle avoidance with pushbroom stereo," *J. Field Robot.*, vol. 35, no. 1, pp. 52–68, 2018.

16. S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2016, pp. 1484–1491.

17. B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017.

18. A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, *Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera*, 2017, pp. 235–252.

19. E. Ackerman, "Skydio demonstrates incredible obstacle-dodging full autonomy with new r1 consumer drone," *IEEE Spectrum*, 2018. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/drones/skydio-r1-drone

20. P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

21. G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *arXiv e-prints*, 2019. [Online]. Available: http://arxiv.org/abs/1904.08405

22. E. Mueggler, B. Huber, and D. Scaramuzza. Event-based, 6-dof pose tracking for high-speed maneuvers using a dynamic vision sensor. Youtube. [Online]. Available: https://youtu.be/LauQ6LWTkxM?t=32

23. A. Mitrokhin, C. Fermuller, C. Parameshwara, and Y. Aloimonos, "Event-based moving object detection and tracking," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

24. T. Stoffregen, G. Gallego, T. Drummond, L. Kleeman, and D. Scaramuzza, "Event-based motion segmentation by motion compensation," in *IEEE International Conference on Computer Vision (ICCV)*, 2019.

25. L. Salt and D. Howard, "Self-adaptive differential evolution for bio-inspired neuromorphic collision avoidance," *CoRR*, vol. abs/1704.04853, 2017. [Online]. Available: http://arxiv.org/abs/1704.04853

26. M. B. Milde, O. J. N. Bertrand, H. Ramachandran, M. Egelhaaf, and E. Chicca, "Spiking elementary motion detector in neuromorphic systems," *Neural Computation*, vol. 30, no. 9, pp. 2384–2417, Sep. 2018.

27. A. Censi, "Efficient neuromorphic optomotor heading regulation," in *American Control Conference (ACC)*, July 2015, pp. 3854–3861.

28. E. Mueller, A. Censi, and E. Frazzoli, "Low-latency heading feedback control with neuromorphic vision sensors using efficient approximated incremental inference," in *IEEE Conf. Decision Control (CDC)*, 2015.

29. A. Glover and C. Bartolozzi, "Event-driven ball detection and gaze fixation in clutter," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2016, pp. 2203–2208.

30. ——, "Robust visual tracking with a freely-moving event camera," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2017, pp. 3769–3776.

31. J. Conradt, R. Berner, M. Cook, and T. Delbruck, "An embedded AER dynamic vision sensor for low-latency pole balancing," in *IEEE Workshop on Embedded Computer Vision (ECV)*, 2009.

32. T. Delbruck and M. Lang, "Robotic goalie with 3ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Front. Neurosci.*, vol. 7, p. 223, 2013.

33. T. Delbruck and P. Lichtsteiner, "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2007, pp. 845–848.

34. X. Clady, C. Clercq, S.-H. Ieng, F. Houseini, M. Randazzo, L. Natale, C. Bartolozzi, and R. Benosman, "Asynchronous visual event-based time-to-contact," *Front. Neurosci.*, vol. 8, no. 9, 2014.

35. F. Galluppi, C. Denk, M. C. Meiner, T. C. Stewart, L. A. Plana, C. Eliasmith, S. Furber, and J. Conradt, "Event-based neural computing on an autonomous mobile platform," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 2862–2867.

36. H. Blum, A. DietmÃijller, M. Milde, J. Conradt, G. Indiveri, and Y. Sandamirskaya, "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor," in *Robotics: Science and Systems (RSS)*, 2017.

37. A. Rosinol Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? combining events, images, and IMU for robust visual SLAM in HDR and high speed scenarios," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 994–1001, Apr. 2018.

38. E. Mueggler, N. Baumli, F. Fontana, and D. Scaramuzza, "Towards evasive maneuvers with quadrotors using dynamic vision sensors," in *Eur. Conf. Mobile Robots (ECMR)*, 2015, pp. 1–8.

39. N. J. Sanket, C. M. Parameshwara, C. D. Singh, A. V. Kuruttukulam, C. Fermuller, D. Scaramuzza, and Y. Aloimonos, "Evdodge: Embodied ai for high-speed dodging on a quadrotor using event cameras," *arXiv e-prints*, 2019. [Online]. Available: http://arxiv.org/abs/1906.02919

40. H. Pomeroy and F. Heppner, "Laboratory determination of startle reaction time of the starling (sturnus vulgaris)," *Animal Behaviour*, vol. 25, pp. 720–725, 1977.

41. Y. Wang and B. J. Frost, "Time to collision is signalled by neurons in the nucleus rotundus of pigeons," *Nature*, vol. 356, no. 6366, p. 236, 1992.

42. B. Cheng, B. W. Tobalske, D. R. Powers, T. L. Hedrick, S. M. Wethington, G. T. Chiu, and X. Deng, "Flight mechanics and control of escape manoeuvres in hummingbirds. i. flight kinematics," *Journal of Experimental Biology*, vol. 219, no. 22, pp. 3518–3531, 2016.

43. I. T. van der Veen and K. M. Lindström, "Escape flights of yellowhammers and greenfinches: more than just physics," *Animal Behaviour*, vol. 59, no. 3, pp. 593–601, 2000.

44. T. S. Otis and W. Gilly, "Jet-propelled escape in the squid loligo opalescens: concerted control by giant and non-giant motor axon pathways." *Proceedings of the National Academy of Sciences*, vol. 87, no. 8, pp. 2911–2915, 1990.

45. R. Batty, "Escape responses of herring larvae to visual stimuli," *Journal of the Marine Biological Association of the United Kingdom*, vol. 69, no. 3, pp. 647–654, 1989.

46. D. Oliva, V. Medan, and D. Tomsic, "Escape behavior and neuronal responses to looming stimuli in the crab chasmagnathus granulatus (decapoda: Grapsidae)," *Journal of Experimental Biology*, vol. 210, no. 5, pp. 865–880, 2007.

47. T. Preuss, P. E. Osei-Bonsu, S. A. Weiss, C. Wang, and D. S. Faber, "Neural representation of object approach in a decision-making motor circuit," *Journal of Neuroscience*, vol. 26, no. 13, pp. 3454–3464, 2006.

48. I. H. Bianco, A. R. Kampff, and F. Engert, "Prey capture behavior evoked by simple visual stimuli in larval zebrafish," *Frontiers in systems neuroscience*, vol. 5, p. 101, 2011.

49. R. A. Ramasamy, B. J. Allan, and M. I. McCormick, "Plasticity of escape responses: prior predator experience enhances escape performance in a coral reef fish," *PLoS One*, vol. 10, no. 8, p. e0132790, 2015.

50. M. H. Holmqvist, "A visually elicited escape response in the fly that does not use the giant fiber pathway," *Visual neuroscience*, vol. 11, no. 6, pp. 1149–1161, 1994.

51. A. Sourakov, "Faster than a flash: the fastest visual startle reflex response is found in a long-legged fly, condylostylus sp.(dolichopodidae)," *The Florida Entomologist*, vol. 94, no. 2, pp. 367–369, 2011.

52. J. D. Sprayberry, "Responses of descending visually-sensitive neurons in the hawkmoth, manduca sexta, to three-dimensional flower-like stimuli." *Journal of Insect Science*, vol. 9, no. 1, 2009.

53. F. T. Muijres, M. J. Elzinga, J. M. Melis, and M. H. Dickinson, "Flies evade looming targets by executing rapid visually directed banked turns," *Science*, vol. 344, no. 6180, pp. 172–177, 2014.

54. E. Burtt and W. Catton, "Visual perception of movement in the locust," *The Journal of physiology*, vol. 125, no. 3, p. 566, 1954.

55. H. L. More and J. M. Donelan, "Scaling of sensorimotor delays in terrestrial mammals," *Proceedings of the Royal Society B: Biological Sciences*, vol. 285, no. 1885, p. 20180613, 2018.

56. D. Woods, J. M Wyma, E. Yund, T. Herron, and B. R. Reed, "Factors influencing the latency of simple reaction time," *Frontiers in human neuroscience*, vol. 9, p. 131, 03 2015.

57. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*. Springer, 1986, pp. 396–404.

58. L. Chittka, P. Skorupski, and N. E. Raine, "Speed–accuracy tradeoffs in animal decision making," *Trends in ecology & evolution*, vol. 24, no. 7, pp. 400–407, 2009.

59. J. E. Mebius, "Derivation of the euler-rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations," *arXiv e-prints*, 2007. [Online]. Available: http://arxiv.org/abs/math/0701759

60. M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, 1996, pp. 226–231.

61. B. Rueckauer and T. Delbruck, "Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor," *Frontiers in neuroscience*, vol. 10, p. 176, 2016.

62. S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *Int. J. Comput. Vis.*, vol. 56, no. 3, pp. 221–255, 2004.

63. A. Fusiello, "Elements of geometric computer vision," http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FUSIELLO4/tutorial.html#x1-130004, University of Edinburgh - School of Informatics, 16.09.2012.

64. R. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, pp. 35–45, 1960.

65. M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Trans. Robot.*, vol. 31, no. 6, pp. 1294–1310, 2015.

66. N. Moshtagh *et al.*, "Minimum volume enclosing ellipsoid," *Convex optimization*, vol. 111, p. 112, 2005.

67. P. Khosla and R. Volpe, "Superquadric artificial potentials for obstacle avoidance and approach," in *icra*, 1988, pp. 1778–1784.

68. D. Eberly, "Distance from a point to an ellipse, an ellipsoid, or a hyperellipsoid," *Geometric Tools, LLC*, 2011.

69. T. Cieslewski, E. Kaufmann, and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2017, pp. 2135–2142.

70. B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Springer Publishing Company, Incorporated, 2016.

71. M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 620–626, Apr. 2018.

72. S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to MAV navigation," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2013.

73. P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer, 2011, vol. 73.

74. T. Delbruck, V. Villanueva, and L. Longinotti, "Integration of dynamic vision sensor with inertial measurement unit for electronically stabilized event-based vision," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2014, pp. 2636–2639.

Acknowledgements: We thank Henri Rebecq and Guillermo Gallego for helpful discussions.
Funding: This work was supported by the SNSF-ERC Starting Grant and the Swiss National Science Foundation through the National Center of Competence in Research (NCCR) Robotics.
Author contributions: The project ideas were conceived by D. Falanga, K. Kleber and D. Scaramuzza. The experiments were designed and performed by D. Falanga and K. Kleber. The paper was written by D. Falanga, K. Kleber and D. Scaramuzza.
Competing interests: The authors declare that they have no competing interests.
Data and materials availability: All (other) data needed to evaluate the conclusions in the paper are present in the paper or the Supplementary Materials.

Supplementary Materials

Time Statistics of Events to Detect Moving Obstacles

To provide an intuitive example of how and why our algorithm successfully classifies static and dynamic events, Fig. S1 shows the simplified case of a mono-dimensional event camera (i.e., an event camera having only one row), rotating in a plane while observing both a static and a dynamic object. The dynamic object (in red) moves from left to right, while the event camera rotates counter-clockwise.

In the center of the figure, we consider a time window spanning from an initial time t_1 to a final time t_5, and we discretize this interval into five time instants to visualize the sequence of events generated by both the motion of the camera and the dynamic object. Let us assume that at time t_1, both objects generate an event due to the motion of the camera: the static object fires an event at pixel p_1, the dynamic object at pixel p_2. At time t_2, the motion of the dynamic object causes another event at pixel p_3, while at time t_3 the motion of the camera generates events at pixels p_2 (static) and p_4 (dynamic). The same concept applies to times t_4 and t_5. After collecting all these events, if we motion-compensate them to remove the effects of the motion of the camera, we obtain a situation like the one depicted at the bottom of the center part of the image, where multiple events get back-projected into the same pixel location.

On the right side of the figure, we report the time statistics of the event project into pixels p_1 to p_4, which are the only ones having motion-compensated events. As one can see, the events belonging to the static part of the scene are equally spread across the time window, while the events fired due to the motion of the dynamic object are concentrated either at the beginning, the center, or the end of the window. If we now compute the mean timestamp of all the events falling in each pixel, subtract the mean of all the events, and normalize it by the length of the time window, we obtain a score for each pixel spanning between -1 and 1. We expect events belonging to the static part of the scene to have a score of approximately zero since they contain events spread across the entire window more or less uniformly. On the contrary, events belonging to the dynamic part of the scene have scores that can span between -1 and 1, depending on where they are concentrated within the time window. In particular, the events generated by the dynamic object at the beginning of the window have a score of -1, those fired at the center of the window have a zero score, while those generated at the end of the window have a score of approximately 1. Since we are interested only in the latest position of the dynamic obstacles, we discard non-positive scores, taking into account only events with a score above zero. Fig. S2 confirms the expected pattern in the statistics of the events in a time window on real data: the first row shows the mean timestamp of a region belonging to the static part of a scene, where the histogram clearly highlights an equal distribution of the events across the entire window; the second row shows the same data for a region belonging to the dynamic part of a scene, where the events are concentrated towards the two ends of the time window.

It is important to notice that, for the sake of making this example simple enough, we only considered one type of event (either positive or negative), while in a real case, each object

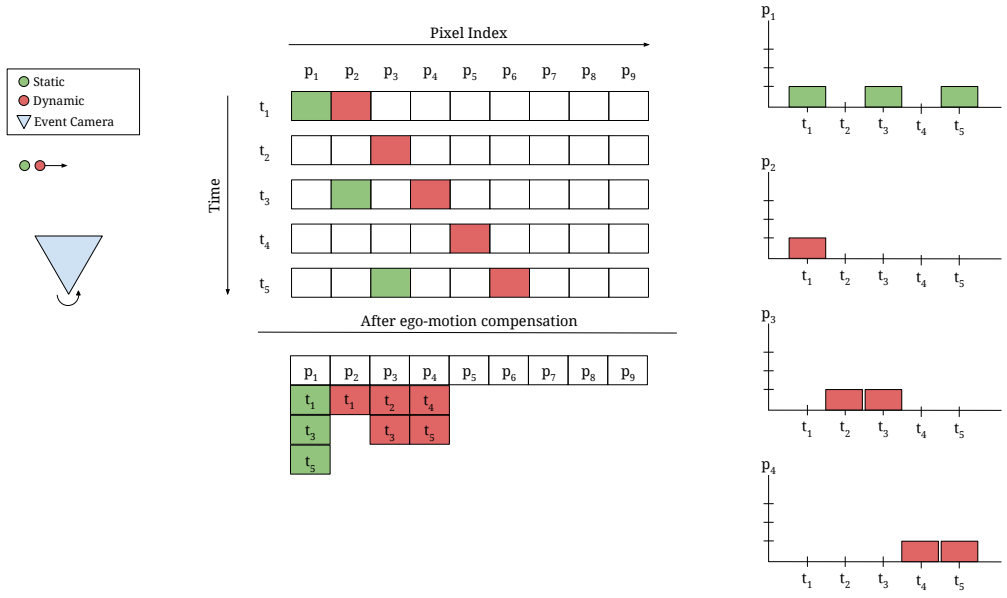


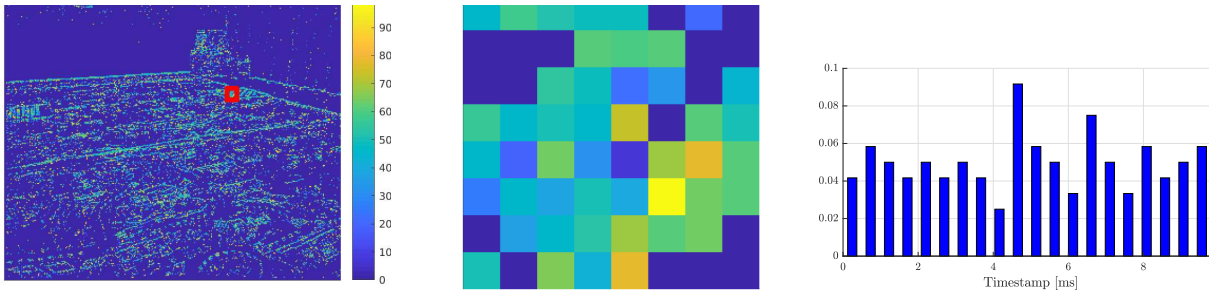
Figure S1: A simple, yet effective example to explain the working principle of our algorithm. On the left side, a one-dimensional event camera is placed in front of two objects: a static object, represented in green, and a dynamic object, in red. The dynamic object moves from left to right, while the event camera rotates in the opposite direction. In the center of the figure, we show on top the sequence of events generated by the motion of the camera and of the dynamic object, for a fixed number of time instant (from an initial time t_1 to a final time t_5), while at the bottom the ego-motion compensated events. On the right side, finally, we report the histogram of the timestamps of all the events falling in each pixel. These histograms clearly highlight the difference in terms of temporal distribution within the time window between the events generated by the static object and the events belonging to the dynamic object.

generated both positive and negative events, simultaneously. However, the principle can be easily extended to events with polarity. Additionally, we invite the reader to notice that not every motion can be compensated, but rather only rotations and roto-translations with respect to a planar scene. Indeed, these motions can be modeled as homographies, and the events they generate can be motion-compensated. However, since we only consider the events that fire within a very short time window, the majority of the scene moves by a very small amount of pixels. Therefore we can approximate the camera motion as a homography. The mathematical description of the motion compensation algorithm is provided in Sec. Ego-Motion Compensation of the Events.

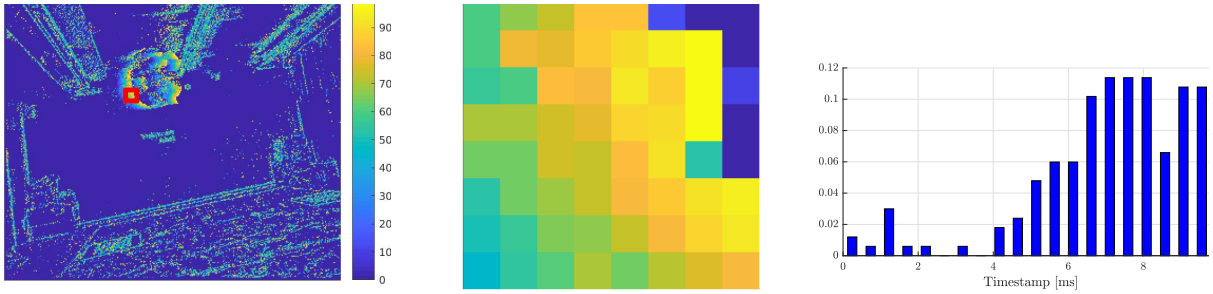
Impact of Using a Simplified Ego-Motion Estimation Algorithm

The ego-motion estimation and compensation algorithm used in our system (Sec. Ego-Motion

Compensation of the Events) assumes the optical flow and, therefore, the events to be only due to the rotation of the camera. In order to account for the linear motion of the camera, one would need some sort of depth estimation, which would render the entire algorithm significantly slower. Nevertheless, using a rotational model to explain the events generated by the ego-motion of the camera is sufficiently accurate to guarantee good overall performance, as shown by the tables in Sec. Accuracy and Success Rate. From a theoretical perspective, this is justified by the fact that translational and rotational optical flow are often very similar and, therefore, very hard to tell apart. As shown in Fig. 15.7 of (73), translation along the X-axis is almost like a rotation around the Y-axis. One needs a long observation time and a smaller focal length (wider field of view) to better distinguish them). Thanks to such a similarity, one can fit a rotational motion to the translational flow to explain the events generated by the latter. This intuition was corroborated in (74), which shows in the supplementary video the impact of ignoring the linear velocity of the motion. This video was generated using only the IMU, without any depth estimation: one can see that the motion-compensated images obtained when the IMU data is taken into account look quite sharp everywhere on the image plane.



(a) A scene without moving objects. The patch highlighted in red in the left mean timestamp image belongs to a static part of the scene, and is reported in the center figure. On the right side, we show the histogram of all the ego-motion events belonging in such patch.



(b) A scene with one moving object. In this case, we selected a patch belonging to a dynamic part of the scene, namely a ball thrown through the field of view of the camera and moving from left to right in the frame. As one can notice, several pixels report a high mean timestamp, and the histogram of all the ego-motion compensated events belonging to the patch confirms this trend.

Figure S2: A figure reporting the statistics of the events within a single time window for two cases: no dynamic object in the scene (top row) and one dynamic object in the scene (bottom row). For each row, we report: on the left, the mean timestamp image, with color-code shown on the right side representing the mean of the timestamps of all the events back-projected to each pixel location; in the center, a $4 \times 4\,\mathrm{pxl}$ patch belonging to a static part (top row) or dynamic part (bottom row) of the scene, taken from the region highlighted in red in the mean timestamp image; on the right, the distribution of the events belonging to that patch. As one can notice, the events in a patch belonging to the static part of the scene report a fairly uniform distribution of their timestamps within the window. Conversely, the events belonging to a dynamic object are very concentrated towards one side of the window (in this case, the end).

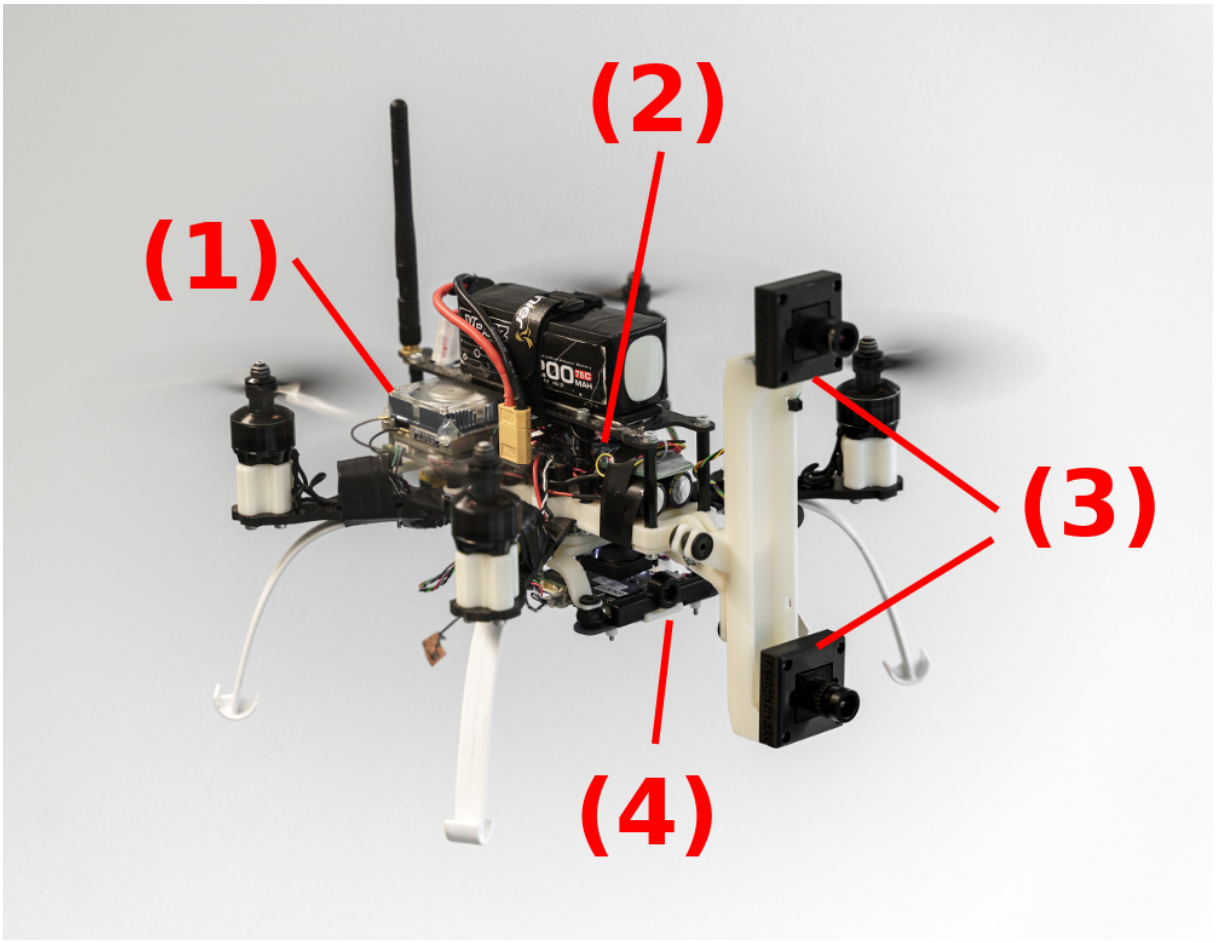


Figure S3: The quadrotor platform we used in our outdoor experiments. The following components are highlighted in the picture: (1) the Nvidia Jetson TX2, running the obstacle detection and avoidance algorithm, as well as the high-level controller; (2) the Lumenier F4 AIO Flight Controller; (3) the two Insightness SEES1 cameras, in a vertical stereo setup; (4) the Qualcomm Snapdragon Flight board, used for state estimation.

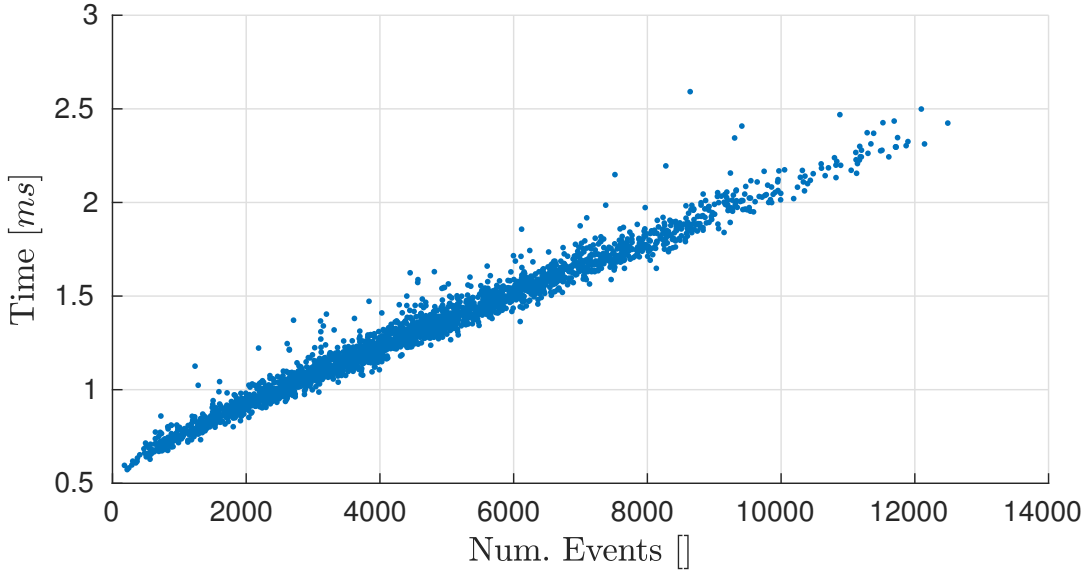


Figure S4: Time necessary to perform the ego-motion compensation as a function of the number of events generated.

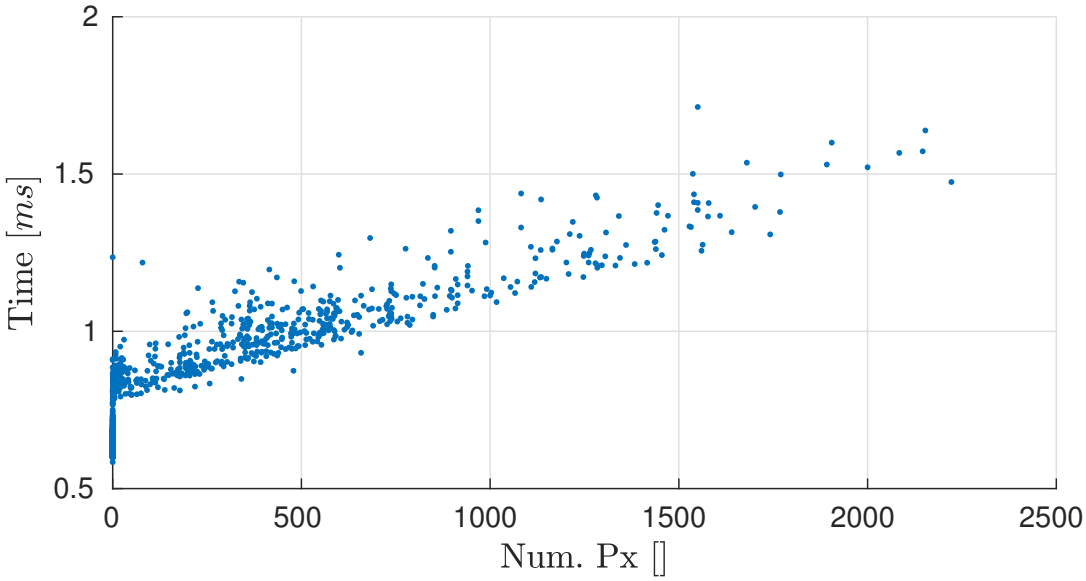
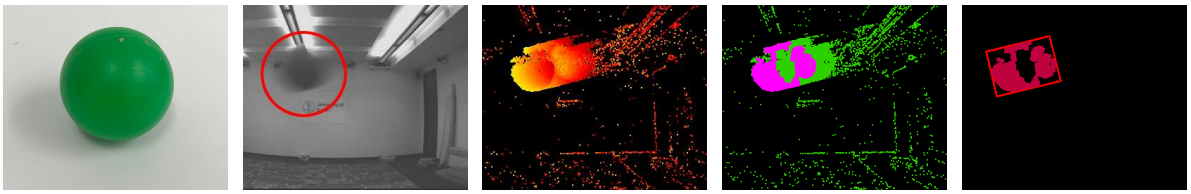
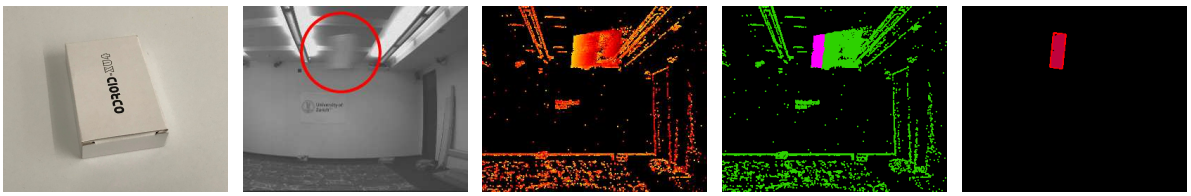


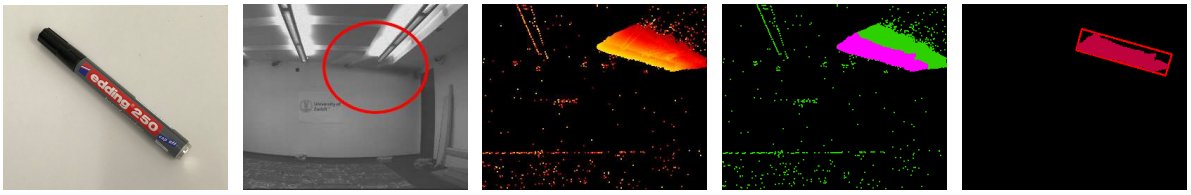
Figure S5: Time necessary to perform the clustering of the scene's dynamic part, depending on the amount of pixels belonging to moving objects.



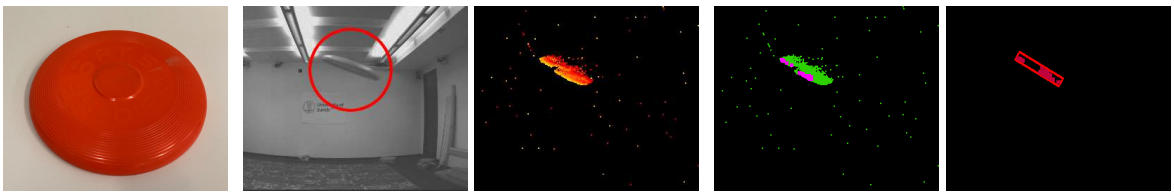
(a) A small-sized ball (radius 4 cm).



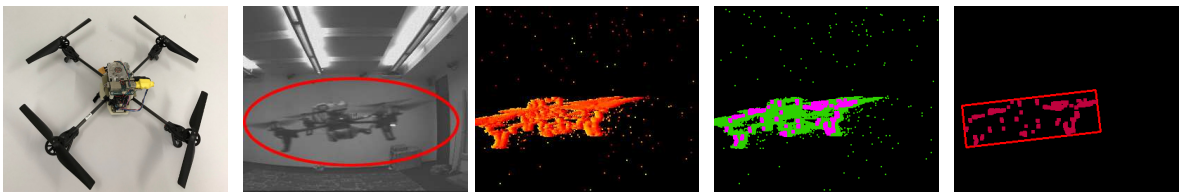
(b) A small-sized marker (width 6 cm, height 9.5 cm, thickness 2.5 cm).



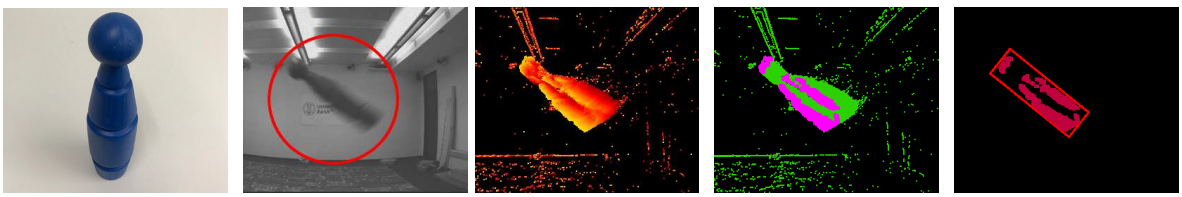
(c) A whiteboard marker (length 14 cm, thickness 1.5 cm).



(d) A frisbee (radius 13.5 cm, height 3.5 cm).

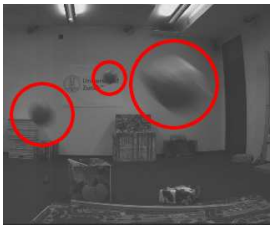


(e) A drone (tip-to-tip diagonal 60 cm, height 10 cm).

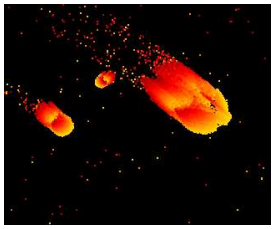


(f) A bowling pin (length 25 cm, thickness 6 cm).

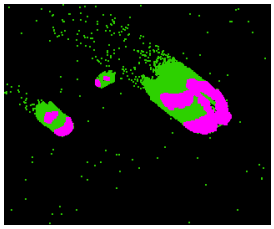
Figure S6: Our algorithm is able to detect different kinds of objects, as shown in this figure. Each row shows the detection of different objects, depicted in the pictures in the first column. From top to bottom: a small-sized ball, a box, a whiteboard marker, a frisbee, a quadrotor, and a bowling pin. These objects were detected using our stereo setup, without any prior information about their shape or size. As one can notice, the frame provided by the on-board camera (second column) presents some motion blur due to the speed of the object, which however is not a problem for our event-based detection algorithm (last column).



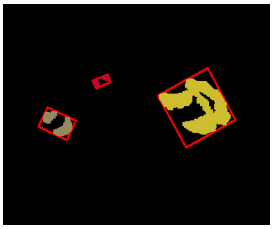
(a) A frame from the on-board Insightness SEES1 camera. The three circles highlight the dynamic obstacles in the scene (three balls of different sizes).



(b) The normalized mean timestamp image generated using the events accumulated in the last time window.



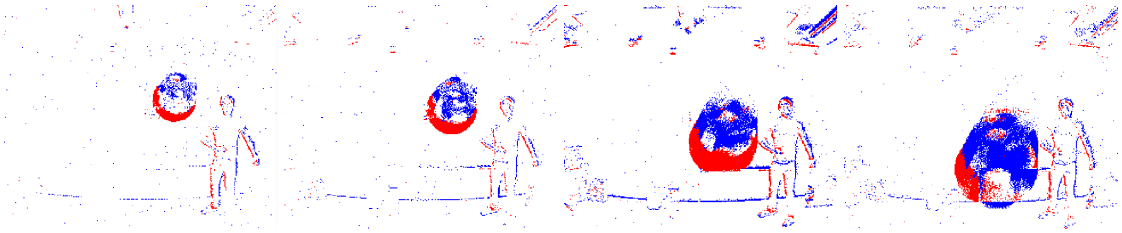
(c) The normalized mean timestamp image after thresholding.



(d) Clustering of the three dynamic obstacles present in the scene.

Figure S7: An example of our algorithm detecting and clustering multiple moving obstacles. (S7a) The frame from the on-board camera, where three moving obstacles, manually circled in red, are visible. (S7b) The mean-timestampe image. (S7c) The mean-timestamp image after thresholding: green represents the static part of the scene, purple indicates events belonging to dynamic obstacles. (S7d) Clustering of the events belonging to different dynamic obstacles.

(a) Frame at time t_0. (b) Frame at time t_1. (c) Frame at time t_2. (d) Frame at time t_3.



(e) Events at time t_0. (f) Events at time t_1. (g) Events at time t_2. (h) Events at time t_3.

(i) Events belonging to moving obstacles at time t_0. (j) Events belonging to moving obstacles at time t_1. (k) Events belonging to moving obstacles at time t_2. (l) Events belonging to moving obstacles at time t_3.

Figure S8: A sequence captured during one of our experiments, where the quadrotor is hovering indoors and an object is thrown towards it with the purpose of evaluating the sensing pipeline. Each column represents a different time, more specifically: $t_0 = 0\,\mathrm{s}$ (first column), $t_1 = 0.05\,\mathrm{s}$ (second column), $t_2 = 0.10\,\mathrm{s}$ (third column), $t_3 = 0.15\,\mathrm{s}$ (fourth column). The first row reports the frame captured by the on-board camera. The second row shows the events, generated by both the motion of the vehicle and the moving obstacle, collected within the last time window of size $\delta t = 10\,\mathrm{ms}$, where blue represents positive events, and red represents negative events. The third row shows the ego-motion compensated events belonging only to the dynamic part of the scene, obtained applying the algorithm described in Sec. Ego-Motion Compensation of the Events.

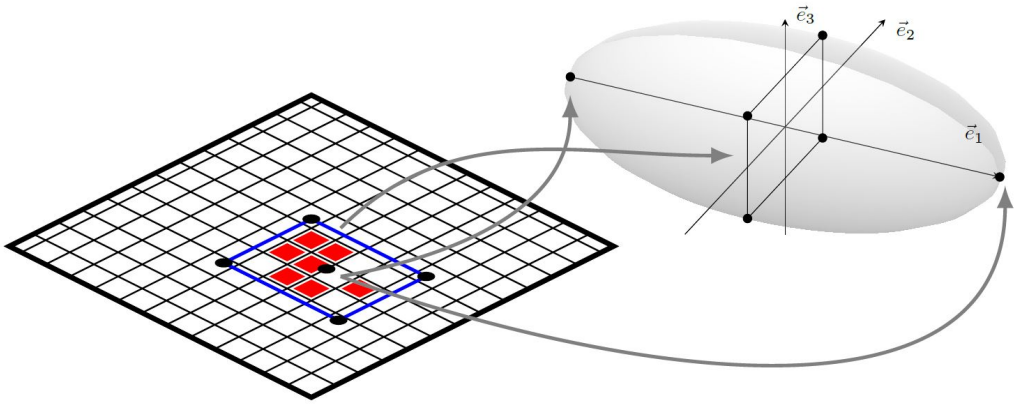
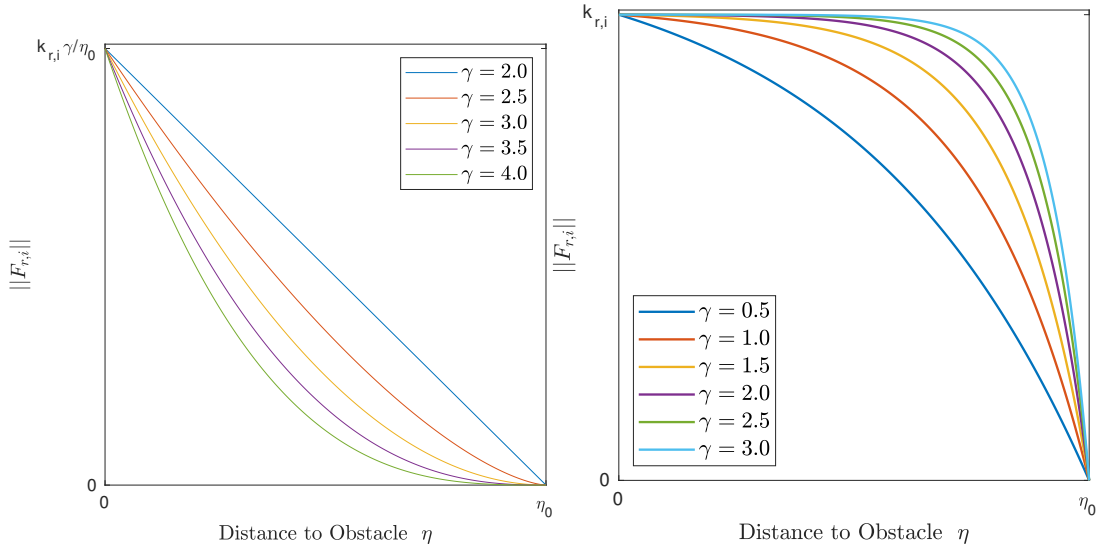


Figure S9: Construction of the obstacle's ellipsoid in the world's frame of reference from the clustered data in the image plane. A minimal volume ellipsoid is fitted around the six projected points using an iterative approach.



(a) Repulsive force as described by (67).

(b) Our proposed repulsive force.

Figure S10: Plots illustrating the two different types of repulsive forces described in this work.

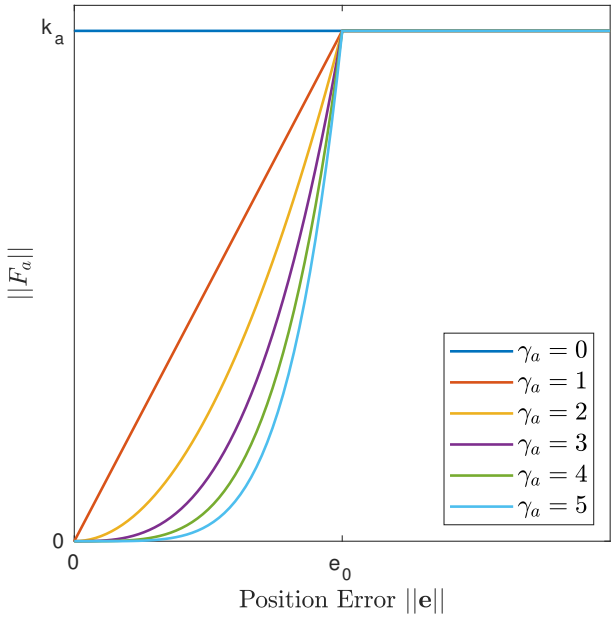


Figure S11: Illustration of the attractive force for different values of γ_a.